资深爬虫工程师专业奉献

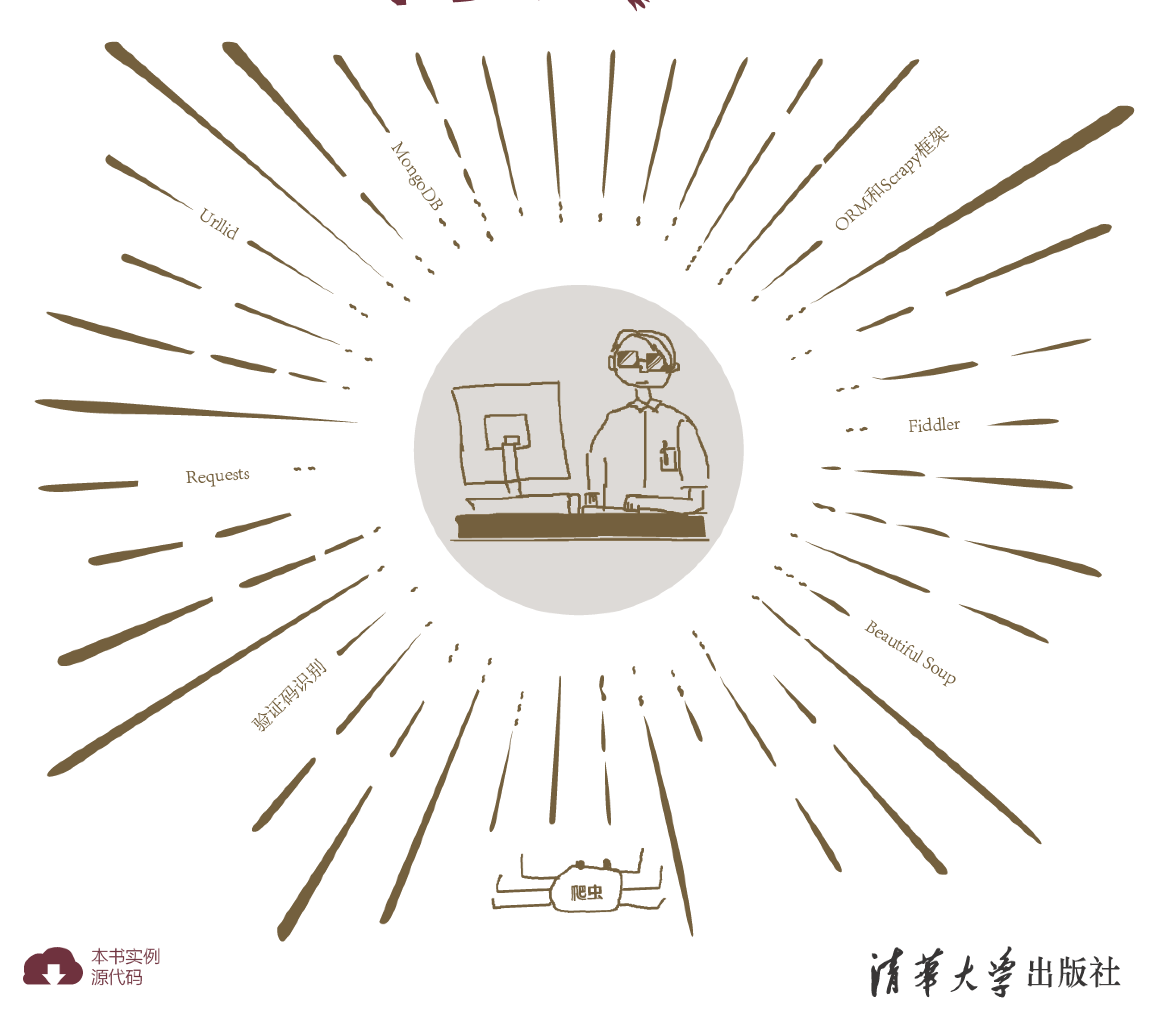
黄永祥 著

基于Python 3编撰

从零基础到项目实战

提供技术交流与服务群

玩转Python网络爬虫



本书实例源代码

清华大学出版社



黄永祥 著

清華大學出版社
北 京

内 容 简 介

本书站在初学者的角度，从原理到实践，循序渐进地讲述了使用Python开发网络爬虫的核心技术。全书从逻辑上可分为基础篇、实战篇和爬虫框架篇三部分。基础篇主要介绍了编写网络爬虫所需的基础知识，分别是网站分析、数据抓取、数据清洗和数据入库。网站分析讲述如何使用Chrome和Fiddler抓包工具对网络做全面分析；数据抓取介绍了Python爬虫模块Urllib和Requests的基础知识；数据清洗主要介绍字符串操作、正则和Beautiful Soup的使用；数据入库分别讲述了MySQL和MongoDB的操作，通过ORM框架SQLAlchemy实现数据持久化，实现企业级开发。实战篇深入讲解了分布式爬虫、爬虫软件开发与应用、12306抢票程序和微博爬取，所举示例均来自于开发实践，可帮助读者快速提升技能，开发实际项目。框架篇主要讲述Scrapy的基础知识，并通过爬取QQ音乐为实例，让读者深层次了解Scrapy的使用。

本书内容丰富，注重实战，适用于从零开始学习网络爬虫的初学者，或者是已经有一些网络爬虫编写经验，但希望更加全面、深入理解Python爬虫的开发人员。

图书在版编目（CIP）数据

玩转Python网络爬虫/黄永祥著．—北京：清华大学出版社，2018
ISBN 978-7-302-50328-6

Ⅰ.①玩… Ⅱ.①黄… Ⅲ.①软件工具—程序设计 Ⅳ.①TP311.56

中国版本图书馆CIP数据核字（2018）第114988号

责任编辑： 王金柱
封面设计： 王　翔
责任校对： 闫秀华
责任印制： 沈　露
出版发行： 清华大学出版社
　网　　址： http://www.tup.com.cn，http://www.wqbook.com
　地　　址： 北京清华大学学研大厦A座　　**邮　　编：** 100084
　社 总 机： 010-62770175　　**邮　　购：** 010-62786544
　投稿与读者服务： 010-62776969，c-service@tup.tsinghua.edu.cn
　质量反馈： 010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者： 北京富博印刷有限公司
装 订 者： 北京市密云县京文制本装订厂
经　　销： 全国新华书店
开　　本： 180mm×230mm　　**印　　张：** 20.25　　**字　　数：** 454千字
版　　次： 2018年8月第1版　　**印　　次：** 2018年8月第1次印刷
印　　数： 1～3500
定　　价： 69.00元

产品编号： 077679-01

前　言

随着大数据和人工智能的普及，Python 的地位也变得水涨船高，许多技术人员投身于 Python 开发，其中网络爬虫是 Python 最为热门的应用领域之一。在爬虫领域，Python 可以说是处于霸主地位，Python 能解决爬虫开发过程中所遇到的难题，开发速度快且支持异步编程，大大缩短了开发周期。因此，Python 爬虫编程已成为爬虫工程师的必备技能。

本书结构

本书共分 18 章，各章内容概述如下：

第 1 章介绍什么是网络爬虫、爬虫的类型和原理、爬虫搜索策略和反爬虫技术以及解决方案。

第 2 章讲解爬虫开发的基础知识，包括 HTTP 协议、请求头和 Cookies 的作用、HTML 的布局结构、JavaScript 的介绍、JSON 的数据格式和 Ajax 的原理。

第 3 章介绍使用 Chrome 开发工具分析爬取网站，重点介绍开发者工具的 Elements 和 Network 标签的功能和使用方式，并通过开发者工具分析 QQ 网站。

第 4 章主要介绍 Fiddler 抓包工具的原理和安装配置，Fiddler 用户界面的各个功能及使用方法。

第 5 章讲述了 Urllib 在 Python 2 和 Python 3 的变化及使用，包括发送请求、使用代理 IP、Cookies 的读写、HTTP 证书验收和数据处理。

第 6 章介绍 Python 第三方库 Requests 的安装和使用，包括发送请求、使用代理 IP、Cookies 的读写、HTTP 证书验收和文件下载与上传。

第 7 章介绍验证码的种类和识别方法，包括 OCR 的安装和使用、验证码图片处理和使用第三方平台识别验证码。

第 8 章讲述数据清洗的三种方法，包括字符串操作（截取、查找、分割和替换）、正则表达式的使用和第三方库 Beautiful Soup 的安装以及使用。

第 9 章讲述如何将数据存储到文件，分别介绍了 CSV、Excel 和 Word 的读写方法及数据存储。

第 10 章介绍 ORM 框架 SQLAlchemy 的安装及使用，实现关系型数据库持久化存储数据，这是企业级的关系型数据库操作。

第 11 章讲述非关系型数据库 MongoDB 的操作，介绍 MongoDB 的安装、原理结构和 Python 实现 MongoDB 读写。

第 12 章介绍爬取淘宝商品信息实例，包括网站分析、数据抓取、数据清洗以及存储在 CSV 文件中，读者应掌握爬虫的开发流程。

第 13 章介绍爬取 QQ 音乐全站歌曲实例，包括网站分析、数据抓取和实现 SQLAlchemy 存储歌曲信息并下载文件，使用异步编程实现分布式开发，提高爬取效率。

第 14 章是在第 12 章的基础上实现爬虫软件开发，包括 PyQt5 的安装、使用 Qt Designer 设计软件界面、搭建 MVC 开发架构。

第 15 章实现 12306 抢票爬虫开发，包括用户登录、查询车次、预订车票、提交订单和生成订单的分析以及功能实现。

第 16 章介绍微博爬虫开发，包括微博登录、采集热门微博、发布微博、关注微博用户和转发评论的分析以及功能实现。

第 17 章介绍 Scrapy 爬虫框架，包括 Scrapy 的运行机制、安装、项目创建以及各个组件的编写（Setting、Items、Item Pipelines 和 Spider）和文件下载。

第 18 章介绍 Scrapy 爬取 QQ 音乐全站歌曲实例，包括编写 Spider 实现数据抓取、Item Pipelines 实现歌曲信息存储和歌曲下载、Items 定义数据存储对象和 Setting 配置项目设置。

本书特色

循序渐进，知识全面：本书站在初学者的角度，围绕 Python 网络爬虫开发展开讲解，从初学者必备基础知识着手，循序渐进地介绍了使用 Python 3 开发网络爬虫的各种知识，内容难度适中，由浅入深，实用性强，覆盖面广，条理清晰，且具有较强的逻辑性和系统性。

实例丰富，扩展性强：本书采用大量的实例进行讲解，力求通过实际操作使读者更容易地掌握爬虫开发。本书实例都经过作者精心设计和挑选，根据作者的实际开发经验总结而来，涵盖了在实际开发中所遇到的各种问题。对于精选案例，都尽可能做到步骤详尽、结构清晰、分析深入浅出，而且案例的扩展性强，读者可根据实际需求扩展开发。

基于理论，注重实践：在讲解过程中，不仅介绍理论知识，而且安排了综合应用实例或小型应用程序，将理论应用到实践中，加强读者的实际开发能力，巩固开发技能和相关知识。

源码提供

本书的实例源码可以在百度网盘下载。

链接地址 1：https://pan.baidu.com/s/1htxBpic　密码：aesy

链接地址 2：https://pan.baidu.com/s/1E7axRN9rC0i9ASM1124lAw

也可以扫描以下二维码下载。



如果你在下载过程中遇到问题，可发送邮件至 booksaga@126.com 获得帮助，邮件标题为“玩转 Python 网络爬虫下载资源”。

技术服务

读者在学习或者工作的过程中，如果遇到实际问题，可以加入 QQ 群 93314951 或 657341423 与笔者联系，笔者会在第一时间给予回复。

读者对象

本书主要适合以下读者阅读：

- Python 网络爬虫初学者及在校学生。
- Python 初级爬虫工程师。
- 从事数据抓取的技术人员。
- 其他学习 Python 网络爬虫的开发人员。

虽然笔者力求本书更臻完美，但由于水平所限，难免会出现错误，特别是实例中爬取的网站可能随时更新，导致源码在运行过程中出现问题，欢迎广大读者和高手专家给予指正，笔者将十分感谢。

编 者

2018 年 1 月

目　录

第 1 章 理解网络爬虫

1.1 爬虫的定义

网络爬虫是一种按照一定的规则自动地抓取网络信息的程序或者脚本。简单来说，网络爬虫就是根据一定的算法实现编程开发，主要通过 URL 实现数据的抓取和发掘。

随着大数据时代的发展，数据规模越来越庞大，数据类型繁多，但是数据价值普遍较低。为了从庞大的数据体系里获取有价值的数据，从而延伸了网络爬虫、数据分析等多个职位。近几年，网络爬虫的需求更是井喷式地爆发，在招聘的供求市场上往往是供不应求，造成这个现状的主要原因就是求职者的专业水平低于需求企业的要求。

传统的爬虫有百度、Google、必应等搜索引擎，这类通用的搜索引擎都有自己的核心算法。但是，这类通用的搜索引擎也存在着一定的局限性：

（1）不同的搜索引擎对于同一个搜索会有不同的结果，搜索出来的结果未必是用户需要的信息。

（2）通用的引擎扩大网络覆盖率，但有限的搜索引擎服务器资源与无限的网络数据资源之间的矛盾将进一步加深。

（3）随着网络上数据形式繁多和网络技术不断发展，图片、数据库、音频、视频多媒体等不同数据大量出现，通用搜索引擎往往对这些信息含量密集且具有一定结构的数据无能为力，不能很好地发现和获取。

因此，为了得到准确的数据，定向抓取相关网页资源的聚焦爬虫应运而生。聚焦爬虫是一个自动下载网页的程序，根据设定的抓取目标有目的性地访问互联网上的网页与相关的 URL，从而获取所需要的信息。与通用爬虫不同，聚焦爬虫并不追求全面的覆盖率，而是抓取与某一特定内容相关的网页，为面向特定的用户提供准备数据资源。

1.2 爬虫的类型

网络爬虫根据系统结构和开发技术大致可以分为 4 种类型：通用网络爬虫、聚焦网络爬虫、增量式网络爬虫和深层网络爬虫。

通用网络爬虫又称全网爬虫，常见的有百度、Google、必应等搜索引擎，爬行对象从一些初始 URL 扩充到整个网站，主要为门户站点搜索引擎和大型网站服务采集数据，具有以下特点：

（1）由于商业原因，引擎的算法是不会对外公布的。

（2）这类网络爬虫的爬行范围和数量巨大，对于爬行速度和存储空间要求较高，爬行页面的顺序要求相对较低。

（3）待刷新的页面太多，通常采用并行工作方式，但需要较长时间才能刷新一次页面。

（4）存在一定缺陷，通用网络爬虫适用于为搜索引擎搜索广泛的需求。

聚焦网络爬虫又称主题网络爬虫，是选择性地爬行根据需求的主题相关页面的网络爬虫。与通用网络爬虫相比，聚焦爬虫只需要爬行与主题相关的页面，不需要广泛地覆盖无关的网页，很好地满足一些特定人群对特定领域信息的需求。

增量式网络爬虫是指对已下载网页采取增量式更新和只爬行新产生或者已经发生变化的网页的爬虫，它能够在一定程度上保证所爬行的页面尽可能是新的页面。只会在需要的时候爬行新产生或发生更新的页面，并不重新下载没有发生变化的页面，可有效减少数据下载量，及时更新已爬行的网页，减小时间和空间上的耗费，但是增加了爬行算法的复杂度和实现难度，基本上这类爬虫在实际开发中不太普及。

深层网络爬虫是大部分内容不能通过静态 URL 获取的、隐藏在搜索表单后的、只有用户提交一些关键词才能获得的网络页面。例如某些网站需要用户登录或者通过提交表单实现提交数据。这类爬虫也是本书主要讲述的重点之一。

这 4 种类型的爬虫大致上又可以分为两类，就是通用爬虫和聚焦爬虫，其中聚焦网络爬虫、增量式网络爬虫和深层网络爬虫可以通俗地归纳为一类，因为这类爬虫都是定向爬取数据。相比于通用爬虫，这类爬虫比较有目的性，也就是网络上经常说的网络爬虫，而通用爬虫在网络上通常称为搜索引擎。

1.3 爬虫的原理

通用网络爬虫的实现原理及过程如图 1-1 所示。

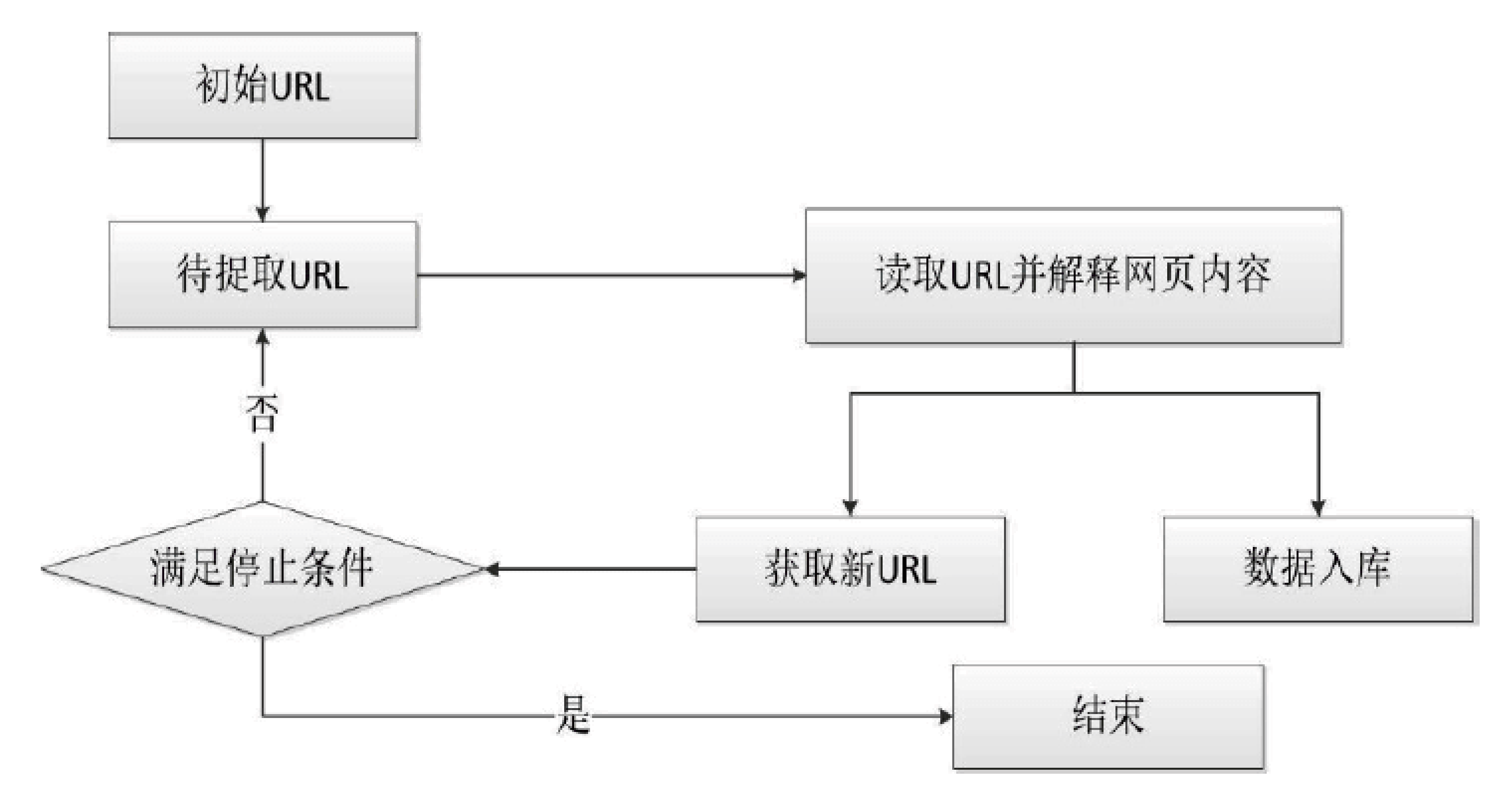


图 1-1　通用爬虫实现的原理及过程

通用网络爬虫的实现原理：

（1）获取初始的 URL。初始的 URL 地址可以人为地指定，也可以由用户指定的某个或某几个初始爬取网页决定。

（2）根据初始的 URL 爬取页面并获得新的 URL。获得初始的 URL 地址之后，先爬取当前 URL 地址中的网页信息，然后解析网页信息内容，将网页存储到原始数据库中，并且在当前获得的网页信息里发现新的URL地址，存放于一个URL队列里面。

（3）从 URL 队列中读取新的 URL，从而获得新的网页信息，同时在新网页中获取新 URL，并重复上述的爬取过程。

（4）满足爬虫系统设置的停止条件时，停止爬取。在编写爬虫的时候，一般会设置相应的停止条件，爬虫则会在停止条件满足时停止爬取。如果没有设置停止条件，爬虫就会一直爬取下去，一直到无法获取新的 URL 地址为止。

聚焦网络爬虫的执行原理和过程与通用爬虫大致相同，在通用爬虫的基础上增加两个步骤：定义爬取目标和筛选过滤 URL，原理如图 1-2 所示。

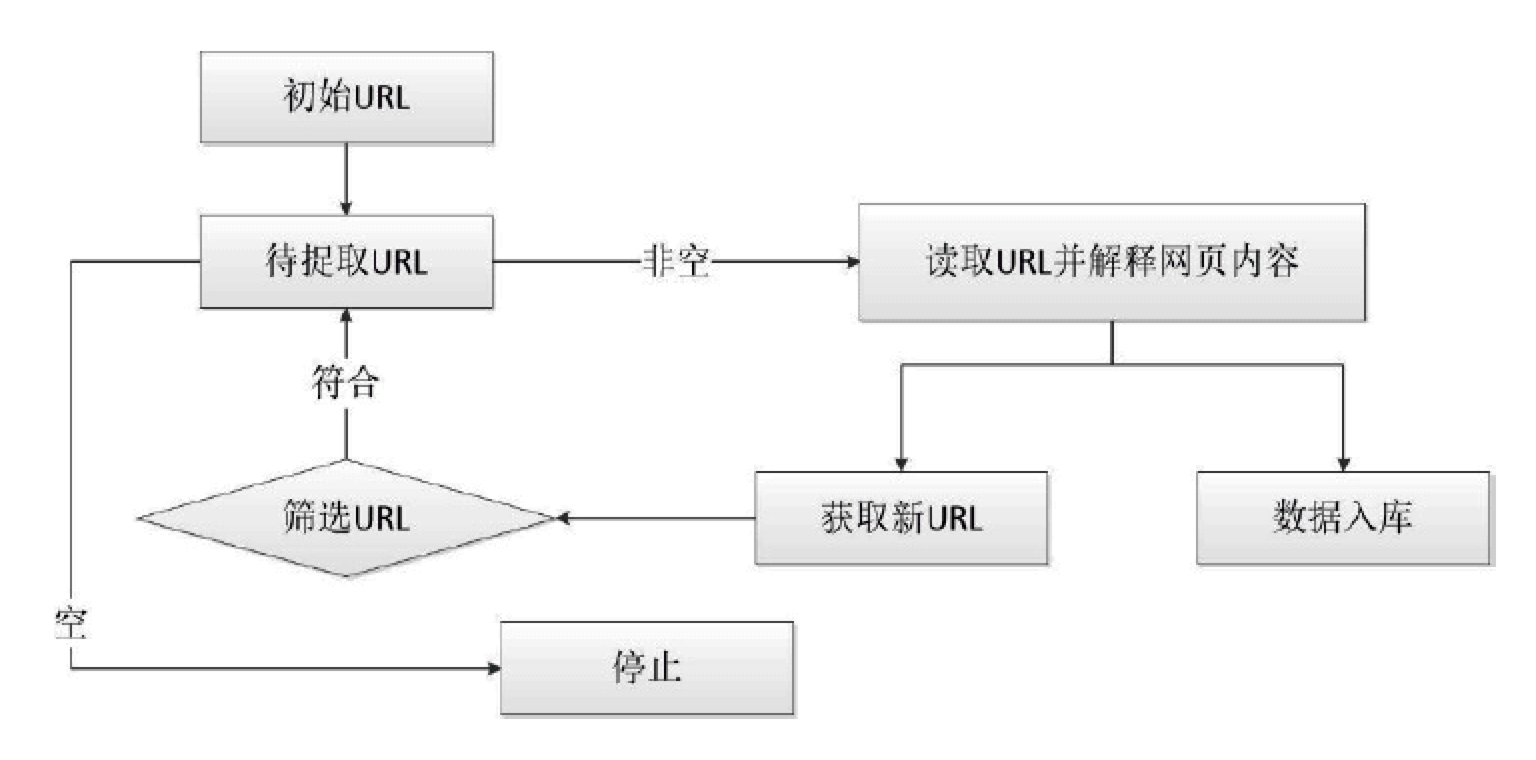


图 1-2 聚焦网络爬虫的原理

聚焦网络爬虫的实现原理：

（1）制定爬取的方案。在聚焦网络爬虫中，首先要依据需求定义聚焦网络爬虫爬取的目标以及整体的爬取方案。

（2）设定初始的 URL。

（3）根据初始的 URL 抓取页面，并获得新的 URL。

（4）从新的 URL 中过滤掉与需求无关的 URL，将过滤后的 URL 放到 URL 队列中。

（5）在 URL 队列中，根据搜索算法确定 URL 的优先级，并确定下一步要爬取的 URL 地址。因为聚焦网络爬虫具有目的性，所以 URL 的爬取顺序不同会导致爬虫的执行效率不同。

（6）得到新的 URL，将新的 URL 重现上述爬取过程。

（7）满足系统中设置的停止条件或无法获取新的 URL 地址时，停止爬行。

1.4 爬虫的搜索策略

在互联网数据时代，有三大搜索策略需要有所了解，下面一一介绍。

1. 深度优先搜索

深度优先搜索是在开发爬虫早期使用较多的方法，目的是达到被搜索结构的叶结点（那些不包含任何超级 URL 的 HTML 文件）。在一个 HTML 文件中，当一个 URL 被选择后，被选 URL 将执行深度优先搜索，搜索后得到新的 HTML 文件，再从新的 HTML 获取新的 URL 进行搜索，以此类推，不断地爬取 HTML 中的 URL，直到 HTML 中没有 URL 为止。

深度优先搜索沿着 HTML 文件中的 URL 走到不能再深入为止，然后返回到某一个 HTML 文件，再继续选择该 HTML 文件中的其他 URL。当不再有其他 URL 可选择时，说明搜索已经结束。其优点是能遍历一个 Web 站点或深层嵌套的文档集合。缺点是因为 Web 结构相当深，有可能造成一旦进去再也出不来的情况发生。

举个例子，比如一个网站的首页里面带有很多 URL，深度优先通过首页的 URL 进入新的页面，然后通过这个页面里的 URL 再进入新的 URL，不断地循环下去，直到返回的页面没有 URL 为止。如果首页有两个 URL，选择第一个 URL 后，生成新的页面就不会返回首页，而是在新的页面选择一个新的 URL，这样不停地访问下去。

2. 宽度优先搜索

宽度优先搜索是搜索完一个 Web 页面中所有的 URL，然后继续搜索下一层，直到底层为止。例如，首页中有 3 个 URL，爬虫会选择其中之一，处理相应的页面之后，然后返回首页再爬取第二个 URL，处理相应的页面，最后返回首页爬取第三个 URL，处理第三个 URL 对应的页面。

一旦一层上的所有 URL 都被选择过，就可以开始在刚才处理过的页面中搜索其余的 URL，这就保证了对浅层的优先处理。当遇到一个无穷尽的深层分支时，不会导致陷进深层文档中出不来的情况发生。宽度优先搜索策略还有一个优点，能够在两个页面之间找到最短路径。

宽度优先搜索策略通常是实现爬虫的最佳策略，因为它容易实现，而且具备大多数期望的功能。但是如果要遍历一个指定的站点或者深层嵌套的 HTML 文件集，用宽度优先搜索策略就需要花费较长时间才能到达最底层。

3. 聚焦爬虫的爬行策略

聚焦爬虫的爬行策略只跳出某个特定主题的页面，根据“最好优先原则”进行访问，快速、有效地获得更多与主题相关的页面，主要通过内容与 Web 的 URL 结构指导进行页面的抓取。聚焦爬虫会给所下载的页面一个评价分，根据得分排序插入一个队列中。最好下一个搜索对弹出队列的第一个页面进行分析后执行，这种策略保证爬虫能优先跟踪那些最有可能 URL 到目标页面的页面。

决定网络爬虫搜索策略的关键是评价 URL 价值，即 URL 价值的计算方法，不同的价值评价方法计算出的 URL 的价值不同，表现出的 URL 的“重要程度”也不同，从而决定不同的搜索策略。由于 URL 包含于页面之中，而通常具有较高价值的页面包含的 URL 也具有较高价值，因此对 URL 价值的评价有时也转换为对页面价值的评价。

1.5 反爬虫技术及解决方案

不同类型的网站都有不一样的发爬虫机制，判断一个网站是否有反爬虫机制需要根据网站设计架构、数据传输方式和请求方式等各个方面评估。下面列出常用的反爬虫技术。

（1）用户请求的 Headers。

（2）用户操作网站行为。

（3）网站目录数据加载方式。

（4）数据加密。

（5）验证码识别。

网站设置反爬机制不代表不能爬取数据，正如“你有张良计，我有过墙梯”，每种反爬虫机制都有对应的解决方案。

1. 基于用户请求的 Headers

从用户请求的 Headers 反爬虫是最常见的反爬虫策略。很多网站会对 Headers 的 User-Agent 进行检测，还有一部分网站会对 Referer 进行检测（一些资源网站的防盗链就是检测 Referer）。如果遇到了这类反爬虫机制，就可以在爬虫代码中添加 Headers 请求头，将浏览器的请求信息以字典的数据格式写入爬虫的请求头。对于检测 Headers 的反爬虫，在爬虫发送请求中修改或者添加 Headers 就能很好地解决。

2. 基于用户操作网站行为

还有一部分网站是通过检测用户行为来判断用户行为是否合规，例如同一 IP 短时间内多次访问同一页面或者同一账户短时间内多次进行相同操作。网站服务器会根据已设定的判断条件判断访问间隔是否合理，从而达到检测用户行为是否合理。

遇到用户行为判断这种情况，可使用 IP 代理，IP 可以在 IP 代理平台上通过 API 接口获取，每请求几次更换一个 IP，这样就能很容易地绕过第一种反爬虫。

还有一种解决方案是在每次请求间隔几秒后再发送下一次请求。只需在代码里面加一个延时功能就可以简单实现，有些有逻辑漏洞的网站可以通过请求几次、退出登录、重新登录、继续请求来绕过同一账号短时间内不能多次进行相同请求的限制。

3. 基于网站目录数据加载

上述几种情况大多都出现在静态页面，还有一部分网站是由 Ajax 通过访问接口的方式生成数据加载到网页。遇到这样的情况，首先分析网站设计，找到 Ajax 请求，分析具体的请求参数和响应的数据结构及其含义，在爬虫中模拟 Ajax 请求，就能获取所需数据。

4. 基于数据加密

在很多情况下没有想象中那么完美，能够直接模拟 Ajax 请求获取数据固然是极好，但部分网站会把请求的参数加密处理。这种情况可先找到加密代码，加密代码主要是使用 JavaScript 实现的，分析代码的加密方式，然后在爬虫代码中模拟其加密处理，再发送请求。这是最优解决方案，但花费时间较多，难度相对较大。

另一种解决方案是使用 Selenium+PhantomJS 框架（自动化测试技术），调用浏览器内核，利用 Selenium 模拟人为操作网页并触发页面中的 JS 脚本，完整地实现自动化操作网页，数据是从网页上自动获取的。这套框架几乎能绕过大多数反爬虫，因为 PhantomJS 是一个没有界面的浏览器。Selenium+PhantomJS 能解决很多事情，而且

用途很大。但是在爬虫中，不到逼不得已的地步，一般不支持使用这种方式，因为这种方式已经是自动化测试的范畴，有点偏离了爬虫开发。

5. 基于验证码识别

最有效的反爬虫技术就是验证码，目前对复杂的验证码还没有做到很好地识别验证，只能通过第三方平台处理或者 OCR 技术识别。当然，不是说有验证码就不能做爬虫，只是目前在验证码的问题上还没有一个很完美的解决方案。

1.6 本章小结

网络爬虫的类型理论上分为 4 类，但实际上可以分为两大类：通用爬虫和聚焦爬虫。通用爬虫主要有 Google、百度、必应等搜索引擎，主要以核心算法为主导，学习成本相对较高。聚焦爬虫就是定向爬取数据，是有目的性的爬虫，学习成本相对较低。

我们常说的网络爬虫大多数以聚焦爬虫为主，其原理和过程与通用爬虫大致相同，读者在编写爬虫程序的时候，需要以设定的爬虫规则和爬取目标为主导，这样更具较强的目的性。

某些网站为防止恶意爬取，常会采用反爬技术，常用的反爬虫技术主要有以下几种：

（1）用户请求的 Headers。

（2）用户操作网站行为。

（3）网站目录数据加载。

（4）数据加密。

（5）验证码识别。

每一种反爬虫技术都有相对应的解决方案，在分析网站的时候，能从网站的设计结构得知是否设置了反爬虫技术，如请求头、用户登录需要验证码、数据生成方式、请求参数是否加密等，了解这些反爬虫技术对编写有实用价值的爬虫程序也很有意义。

第 2 章

爬虫开发基础

2.1 HTTP 与 HTTPS

HTTP 是一个客户端和服务器端请求和应答的标准（TCP）。客户端是终端用户，服务器端是网站。通过使用 Web 浏览器、网络爬虫或者其他工具，客户端发起一个到服务器上指定端口（默认端口为 80）的 HTTP 请求。这个客户端叫用户代理（User Agent）。响应的服务器上存储着资源，比如 HTML 文件和图像，这个服务器为源服务器（Origin Server），在用户代理和服务器中间可能存在多个中间层，比如代理、网关或者隧道（Tunnels）。

通常，由 HTTP 客户端发起一个请求，建立一个到服务器指定端口（默认是 80 端口）的 TCP 连接，HTTP 服务器则在那个端口监听客户端发送过来的请求，一旦收到请求，服务器（向客户端）发回一个状态行（比如 "HTTP/1.1 200 OK"）和（响应的）消息，

消息的消息体可能是请求的文件、错误消息或者其他一些信息。

在浏览器的地址栏输入的网站地址叫作 URL（Uniform Resource Locator，统一资源定位符）。就像每家每户都有一个门牌地址一样，每个网页也都有一个 Internet 地址。在浏览器的地址框中输入一个 URL 或单击一个超级 URL 时，URL 就确定了要浏览的地址，向服务器发送一次请求，浏览器通过超文本传输协议（HTTP）传送到服务器，服务器根据请求头做出相应的响应，将响应数据返回到客户端，客户端收到响应内容后，通过浏览器翻译成网页。

HTTP 协议传输的数据都是未加密的，也就是明文的数据，因此使用 HTTP 协议传输隐私信息非常不安全。为了保证这些隐私数据能加密传输，于是网景公司设计了 SSL（Secure Sockets Layer）协议用于对 HTTP 协议传输的数据进行加密，从而诞生了 HTTPS。

HTTPS 在传输数据之前需要客户端（浏览器）与服务端（网站）之间进行一次握手，在握手过程中将确立双方加密传输数据的密码信息。TLS/SSL 协议不仅是一套加密传输的协议，更是一件经过艺术家精心设计的艺术品，TLS/SSL 中使用了非对称加密、对称加密以及 HASH 算法。握手过程的简单描述如下：

（1）浏览器将自己支持的一套加密规则发送给网站。

（2）网站从中选出一组加密算法与 HASH 算法，并将自己的身份信息以证书的形式发回给浏览器。证书里面包含网站地址、加密公钥以及证书的颁发机构等信息。

（3）获得网站证书之后浏览器要做以下工作：

① 验证证书的合法性（如颁发证书的机构是否合法、证书中包含的网站地址是否与正在访问的地址一致等），如果证书受信任，浏览器栏就会显示一个小锁头，否则会给出证书不受信任的提示。

② 如果证书受信任或者用户接受了不受信任的证书，浏览器就会生成一串随机数的密码，并用证书中提供的公钥加密。

③ 使用约定好的 HASH 计算握手消息，并使用生成的随机数对消息进行加密，最后将之前生成的所有信息发送给网站。

（4）网站接收浏览器发来的数据之后要做以下操作：

① 使用自己的私钥将信息解密并取出密码，使用密码解密浏览器发来的握手消息，并验证 HASH 是否与浏览器发来的一致。

② 使用密码加密一段握手消息，发送给浏览器。

（5）如果浏览器解密并计算握手消息的 HASH 与服务端发来的 HASH 一致，此时握手过程结束，之后所有的通信数据将使用之前浏览器生成的随机密码，并利用对称加密算法进行加密。

浏览器与网站互相发送加密的握手消息并验证，目的是保证双方都获得一致的密码，并且可以正常地加密、解密数据，为真正数据的传输做一次测试。另外，HTTPS 一般使用的加密与 HASH 算法如下。

（1）非对称加密算法：RSA、DSA/DSS。

（2）对称加密算法：AES、RC4、3DES。

（3）HASH 算法：MD5、SHA1、SHA256。

其中，非对称加密算法用于在握手过程中加密生成的密码，对称加密算法用于对真正传输的数据进行加密，而 HASH 算法用于验证数据的完整性。由于浏览器生成的密码是整个数据加密的关键，因此在传输的时候使用非对称加密算法对其加密。非对称加密算法会生成公钥和私钥，公钥只能用于加密数据，可以随意传输，而网站的私钥用于对数据进行解密，所以网站都会非常小心地保管自己的私钥，防止泄漏。

TLS 握手过程中有任何错误都会使加密连接断开，从而阻止隐私信息的传输。正是由于 HTTPS 非常安全，攻击者无法从中找到下手的地方，因此更多地采用假证书的手法来欺骗客户端，从而获取明文的信息。

2.2 请求头

请求头描述客户端向服务器发送请求时使用的协议类型、所使用的编码以及发送内容的长度等。客户端（浏览器）通过输入 URL 后确定等于做了一次向服务器的请求动作，在这个请求里面带有请求参数，请求头在网络爬虫中的作用是相当重要的一

部分。检测请求头是常见的反爬虫策略，因为服务器会对请求头做一次检测来判断这次请求是人为的还是非人为的。为了形成一个良好的代码编写规范，无论网站是否做 Headers 反爬虫机制，最好每次发送请求都添加请求头。

请求头的参数如下。

（1）Accept：text/html,image/*（浏览器可以接收的类型）。

（2）Accept-Charset：ISO-8859-1（浏览器可以接收的编码类型）。

（3）Accept-Encoding：gzip,compress（浏览器可以接收压缩编码类型）。

（4）Accept-Language：en-us,zh-cn（浏览器可以接收的语言和国家类型）。

（5）Host：请求的主机地址和端口。

（6）If-Modified-Since：Tue, 11 Jul 2000 18:23:51 GMT（某个页面的缓存时间）。

（7）Referer：请求来自于哪个页面的 URL。

（8）User-Agent：Mozilla/4.0（compatible，MSIE 5.5，Windows NT 5.0，浏览器相关信息）。

（9）Cookie：浏览器暂存服务器发送的信息。

（10）Connection：close(1.0)/Keep-Alive(1.1)（HTTP 请求版本的特点）。

（11）Date：Tue, 11 Jul 2000 18:23:51 GMT（请求网站的时间）。

一个标准的请求基本上都带有以上属性。在网络爬虫中，请求头一定要有 User-Agent，其他的属性可以根据实际进行添加，因为反爬虫通常检测请求头的 Referer 和 User-Agent，而 Cookie 不能添加到请求头。除此之外，还有一些比较特殊的请求头信息，如 Upgrade-Insecure-Requests（告诉服务器，浏览器可以处理 HTTPS 协议）、X-Requested-With（判断是否 Ajax 请求）等。

以下是 Python 里面一个完整的请求头，以字典格式生成，代码如下：

```
Headers = {
    'Accept': 'text/html,application/xhtml+xml,
              application/xml;q=0.9 ,*/*;q=0.8',
    'Accept-Language': 'zh-CN,zh;q=0.8',
    'Cache-Control': 'max-age=0',
    'User-Agent': ' Mozilla/5.0 (Windows NT 6.3;
```

```
            WOW64; rv:41.0) Gecko/20100101 Firefox/41.0',
'Connection': 'keep-alive',
    'Referer': 'https://movie.douban.com/'}
```

2.3 Cookies

Cookies 也可以称为 Cookie，指某些网站为了辨别用户身份、进行 Session 跟踪而储存在用户本地终端上的数据。

一个 Cookies 就是存储在用户主机浏览器中的文本文件。Cookies 是纯文本形式，它们不包含任何可执行代码。服务器告诉浏览器将这些信息存储，并且每个请求中都将该信息返回到服务器。服务器之后可以利用这些信息来标识用户。多数需要登录的网站通常会在用户登录后将用户信息写入 Cookies，只要这个 Cookies 存在并且合法，就可以自由地浏览这个网站的所有站点。Cookies 只是包含数据，就其本身而言并不有害。

服务器可以利用 Cookies 包含的信息判断在 HTTP 传输中的状态。Cookies 最典型的应用是判定注册用户是否已经登录网站和保留用户信息以便简化登录手续。

一般 Cookies 所具有的属性如下。

- Domain：域，表示当前 Cookies 属于哪个域或子域下面。
- Path：表示 Cookies 的所属路径。
- Expire Time/Max-Age：表示 Cookies 的有效期。
- Secure：表示该 Cookies 只能用 HTTPS 传输。
- Httponly：表示此 Cookies 必须用 HTTP 或 HTTPS 传输。
- HasKeys：通过该值指示 Cookie 是否含有子键，返回一个 bool 值。
- Name：表示 Cookie 的名称。
- Value：单个 Cookie 的值。
- Values：单个 Cookie 所包含的键值对的集合。

Cookies 的优点如下。

（1）极高的扩展性和可用性。

（2）通过良好地编程控制保存在 Cookie 中的 Session 对象的大小。

（3）通过加密和安全传输技术（SSL）减少 Cookie 被破解的可能性。

（4）只在 Cookie 中存放不敏感数据，即使被盗也不会有重大损失。

（5）可控制 Cookie 的生命期，使之不会永远有效。

Cookies 的缺点如下。

（1）Cookie 数量和长度的限制。每个 domain 最多只能有 20 条 Cookie，每个 Cookie 长度不能超过 4KB，否则会被截掉。

（2）安全性问题。如果 Cookie 被拦截，就有可能被取得所有的 Session 信息。

（3）某些状态不可保存在客户端。例如，为了防止重复提交表单，需要在服务器端保存一个计数器。如果把这个计数器保存在客户端，那么它起不到任何作用。

2.4 HTML

HTML 是超文本标记语言，标准通用标记语言下的一个应用。“超文本”就是指页面内可以包含图片、链接，甚至音乐、程序等非文字元素。超文本标记语言的结构包括“头”部分（Head）和“主体”部分（Body），其中“头”部分提供关于网页的信息，“主体”部分提供网页的具体内容。

爬虫开发对 HTML 的要求是能看懂 HTML 各个标签的含义，了解标签的属性作用以及整个 HTML 布局设计。从一个简单的 HTML 开始了解：

```
<!DOCTYPE html> # 声明为 HTML5 文档
<html># 元素是 HTML 页面的根元素
<head># 元素包含了文档的元（meta）数据
<meta charset="utf-8"># 元素可提供有关页面的元信息（meta-information），主要是描述和关键词
<title>Python</title># 元素描述了文档的标题
</head>
<body> # 元素包含了可见的页面内容
```

```
<h1> 我的第一个标题 </h1>  #  定义一个标题
<p> 我的第一个段落。</p>  #  元素定义一个段落
</body>
</html>
```

一个完整的网页必定以 <html></html> 为开头和结尾，整个 HTML 可分为两部分：

（1）<head></head>，主要是对网页的描述、图片和 JavaScript 的引用。<head> 元素包含所有的头部标签元素。在 <head> 元素中可以插入脚本（scripts）、样式文件（CSS）及各种 meta 信息。该区域可添加的元素标签有 <title>、<style>、<meta>、<link>、<script>、<noscript> 和 <base>。

（2）<body></body> 是网页信息的主要载体。该标签下还可以包含很多类别的标签，不同的标签有不同的作用，标签以 <> 开头，以 </> 结尾，<> 和 </> 之间的内容是标签的值和属性，每个标签之间可以是相互独立的，也可以是嵌套、层层递进的关系。

根据这两个组成部分就能很容易地分析整个网页的布局。其中，<body></body> 是整个 HTML 的重点部分，通过示例讲述如何分析 <body></body>：

```
<body>
<h1> 我的第一个标题 </h1>
<div>
<p> Python</p>
</div>
<h2>
<p>
<a> Python</a>
</p>
</h2>
</body>
```

上述例子分析如下：

（1）<h1> 和 <div> 是两个不相关的标签，两个标签是相互独立的。

（2）<div> 和 <p> 是嵌套关系，<p> 的上一级标签是 <div>。

（3）<h1> 和 <p> 这两个标签是毫无关系的。

（4）<h2> 标签包含一个 <p> 标签，<p> 标签再包含一个 <a> 标签，一个标签可以包含多个标签在其中。

除上述示例的标签之外，大部分标签都可以在 <body></body> 中添加，常用的标签如表 2-1 所示。

表2-1 HTML常用的标签

HTML标签	中文释义
Img	图片
A	锚
Strong	加重（文本）
Em	强调（文本）
I	斜体字
B	粗体（文本）
Br	换行
Div	分隔
Span	范围
Ol	排序列表
Ul	不排序列表
Li	列表项目
Dl	定义列表
h1~h6	标题1到标题6
P	段落
Tr	表格中的一行
Th	表格中的表头
Td	表格中的一个单元格

2.5 JavaScript

JavaScript 是一种直译式脚本语言，是一种动态类型、弱类型、基于原型的语言，内置支持类型。它的解释器被称为 JavaScript 引擎，为浏览器的一部分，广泛用于客户端的脚本语言，最早是在 HTML（标准通用标记语言下的一个应用）网页上使用的，用来给 HTML 网页增加动态功能。

JavaScript 脚本语言同其他语言一样，有自身的基本数据类型、表达式和算术运算符及程序的基本框架。JavaScript 提供了 4 种基本的数据类型和两种特殊数据类型

用来处理数据和文字。而变量提供存放信息的地方，表达式则可以完成较复杂的信息处理。

有时候分析网站需要理解某些 JavaScript 的功能，如某些特殊的数据会存放在 JavaScript 中。以 12306 全国站点为例，如图 2-1 所示。

```
← → C  ▲ 不安全 | https://kyfw.12306.cn/otn/resources/js/framework/station_name.js?station_version=1.9031

var station_names =' @bjb|北京北|VAP|beijingbei|bjb|0@bjd|北京东|BOP|beijingdong|bjd|1@bji|北京|BJP|beijing|bj|2@bjn|北京南|V
南|IZQ|guangzhounan|gzn|5@cqb|重庆北|CUW|chongqingbei|cqb|6@cqi|重庆|CQW|chongqing|cq|7@cqn|重庆南|CRW|chongqingnan|cqn|8@gz
海南|SNH|shanghainan|shn|11@shq|上海虹桥|AOH|shanghaihongqiao|shhq|12@shx|上海西|SXH|shanghaixi|shx|13@tjb|天津北|TBP|tianji
南|TIP|tianjinnan|tjn|16@tjx|天津西|TXP|tianjinxi|tjx|17@cch|长春|CCT|changchun|cc|18@ccn|长春南|CET|changchunnan|ccn|19@ccx
东|ICW|chengdudong|cdd|21@cdn|成都南|CNW|chengdunan|cdn|22@cdu|成都|CDW|chengdu|cd|23@csh|长沙|CSQ|changsha|cs|24@csn|长沙南
南|FYS|fuzhounan|fzn|27@gya|贵阳|GIW|guiyang|gy|28@gzh|广州|GZQ|guangzhou|gz|29@gzx|广州西|GXQ|guangzhouxi|gzx|30@heb|哈尔滨
西|VAB|haerbinxi|hebx|33@hfe|合肥|HFH|hefei|hf|34@hfx|合肥西|HTH|hefeixi|hfx|35@hhd|呼和浩特东|NDC|huhehaotedong|hhhtd|36@hh
东|KEQ|haikoudong|hkd|38@hkd|海口东|HMQ|haikoudong|hkd|39@hko|海口|VUQ|haikou|hk|40@hzd|杭州东|HGH|hangzhoudong|hzd|41@hzh|杭
南|JNK|jinan|jn|44@jnd|济南东|JAK|jinandong|jnd|45@jnx|济南西|JGK|jinanxi|jnx|46@kmi|昆明|KMM|kunming|km|47@kmx|昆明西|KXM|k
东|LVJ|lanzhoudong|lzd|50@lzh|兰州|LZJ|lanzhou|lz|51@lzx|兰州西|LAJ|lanzhouxi|lzx|52@nch|南昌|NCG|nanchang|nc|53@nji|南京|NJ
```

图 2-1 12306 站点信息

从图 2-1 中可以看到，不同的站点有对应的英文字母，代表站点的编码信息，JavaScript 存储数据主要以变量的形式。

JavaScript 还能根据用户触发某些事件对用户的操作进行加工处理。例如用户登录信息设置加密处理，原理是先对用户提交信息做加密处理，然后发送请求到服务器，这一系列事件由 JavaScript 独立完成。要在爬虫实现该功能，就要分析 JavaScript 如何执行整个用户登录过程。

分析一个简单的例子，进一步了解 JavaScript 事件触发原理：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<script> <!-- //JavaScript 代码 -->
function validateForm() {
    var x = document.forms["myForm"]["fname"].value;
    if (x == null || x == "") {
        alert(" 需要输入名字。");
        return false;
    }
    else{alert(" 你的名字提交成功 ")}
}
```

```
</script>
</head>
<body>
<!-- //html 表单，当点击按钮 " 提交 " 后会触发 onsubmit 这个事件，执行
JavaScript 代码 -->
<form name="myForm" action="" onsubmit="return validateForm()"
method="post">
名字 : <input type="text" name="fname">
<input type="submit" value=" 提交 ">
</form>
</body>
</html>
```

JavaScript 事件的触发过程：

（1）HTML 根据 <form></form> 标签相应地生成一个表单。

（2）当用户在表单输入内容后，单击提交按钮，就会触发 <form></form> 表单里所指向 validateForm() 方法，执行相应的 JavaScript 代码。

（3）validateForm() 会判断输入的值是否为空。如果输入的值为空，就提示输入名字；若输入的值不为空，则提示“提交成功”。

2.6 JSON

JSON（JavaScript Object Notation，JS 对象标记）是一种轻量级的数据交换格式，采用完全独立于编程语言的文本格式来存储和表示数据。简洁和清晰的层次结构使得 JSON 成为理想的数据交换语言，易于阅读和编写，同时也易于机器解析和生成，并有效地提升网络传输效率。

在 JS 语言中，一切都是对象。因此，任何支持的类型都可以通过 JSON 来表示，例如字符串、数字、对象、数组等。JSON 格式说明如下：

（1）对象表示为键值对。

（2）数据由逗号分隔。

（3）花括号保存对象。

（4）方括号保存数组。

JSON 的书写格式是：键 / 值对，包括字段名称（字符串），后面写一个冒号，然后是值。例如“name”:“Tom”，等价于 JavaScript 语句：name = “Tom”

JSON 的值可以是数字（整数或浮点数）、字符串、逻辑值（True 或 False）、数组（在方括号中）、对象（在花括号中）和 Null。

例子如下：

```
MyJSon = {
"name": "Python",
"address" :  { "province" :  "广东" , "city" : "广州"}
}
```

JSON 的格式是用花括号表示的，代码 MyJSon 里包含两个属性，分别是 name 和 address。name 的值是“Python”；address 的值是嵌套新的 JSON，里面包含 province 和 city 属性，值为“广东”和“广州”。

一个 JSON 里可以嵌套多个 JSON，也可以嵌套 JSON 数组，都是以键 - 值的形式表现。在数据结构上，JSON 与 Python 里的字典非常相似。

2.7 Ajax

Ajax 不是一种新的编程语言，而是一种用于创建更好、更快以及交互性更强的 Web 应用程序的技术。使用 JavaScript 向服务器提出请求并处理响应而不阻塞用户，核心对象是 XMLHTTPRequest。通过这个对象，JavaScript 可在不重载页面的情况下与 Web 服务器交换数据，即在不需要刷新页面的情况下就可以产生局部刷新的效果。

Ajax 在浏览器与 Web 服务器之间使用异步数据传输（HTTP 请求），这样就可以使网页从服务器请求少量的信息，而不是整个页面。

JavaScript、XML、HTML 与 CSS 在 Ajax 中使用的 Web 标准已被良好定义，并被所有的主流浏览器支持，Ajax 应用程序独立于浏览器和平台。

Web 应用程序比桌面应用程序有优势，能够涉及广大的用户，更易安装及维护，也更易开发。

判断网页数据是否使用 Ajax 最简单的方法：触发事件之后，判断网页是否发生刷新状态。如果网页没有发生刷新，数据就自动生成，说明数据的加载是通过 Ajax 生成并渲染到网页上的；反之，数据是通过服务器后台生成并加载的。

两种数据加载渲染方式分别由前端和后端完成，实现的方式和原理也不同。判断数据加载方式是爬虫开发必备的基本技能之一，正确地判断数据加载方式才能找到数据来源的渠道，最终才能找到抓取的目标。

2.8 本章小结

本章主要介绍了与编写爬虫程序相关的 Web 前端开发技术。

前端开发技术是爬虫开发人员必备技能之一，也是编写爬虫程序的基础。前端技术的主要作用是分析各类网站的设计架构，以便有针对性地编写爬虫脚本。从整个爬虫开发周期来看，分析网站架构是最为耗时的一环，也是爬虫开发的核心之一，可以说，爬虫的开发都是基于网站的分析为前提。

关于前端开发技术，读者应重点掌握以下内容。

- HTTP 与 HTTPS：互联网上应用最为广泛的一种网络协议。目前所有网站开发都基于该协议，也是网站的实现原理。
- 请求头：基于 HTTP 与 HTTPS 协议实现，其作用是在通信之间实现信息传递。熟知各种请求类型，对爬虫中编写请求头有指导性作用。
- Cookies：存储在用户主机浏览器中的文本文件，主要让服务器识别各个用户身份信息。
- HTML：服务器返回的网页内容，一般由服务器后台生成。网站大部分数据来源于此，熟悉 HTML 布局和各个标签的作用，有利于数据抓取和清洗。
- JavaScript：主要实现网页的动态功能以及用户交互。要懂得分析 JavaScript 代码，尤其是数据加密处理。
- JSON：表示一个 JavaScript 对象的信息，本质是一个特殊的字符串。
- Ajax：主要是前端数据加载和渲染技术，其响应内容大部分以 JSON 格式为主。

第 3 章

Chrome 分析网站

3.1 Chrome 开发工具

浏览器是从事编程开发人员必备的开发工具。世界上五大主流浏览器分别是：IE、Opera、Google Chrome、Safari 和 Firefox。其中，Chrome 和 Firefox 是编程开发人员的首选，主要是两者运行速度、扩展性和用户体验都符合开发人员所需。

本书选择 Chrome 作为分析网站的工具，因为其简洁、速度快（无论是启动速度、页面解析速度还是 JavaScript 执行速度），对 HTML5 和 CSS3 的支持也比较完善。

以分析豆瓣电影为例，先打开 Chrome 浏览器，进入豆瓣电影网页（https://movie.douban.com/）。单击 Chrome 的开发者工具（快捷键：F12），如图 3-1 所示。

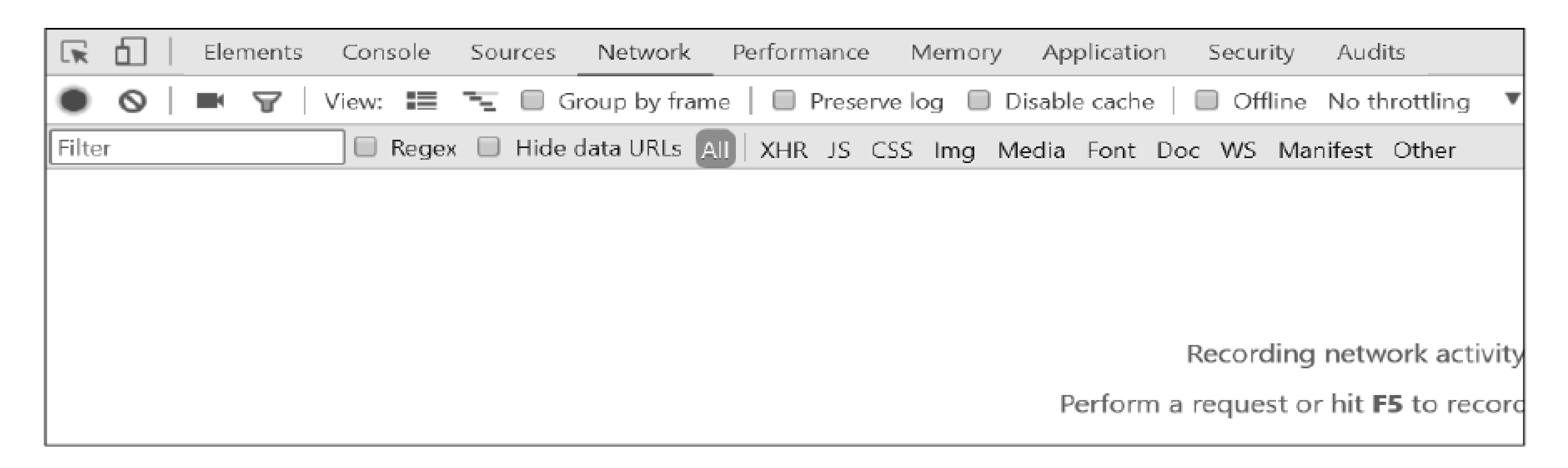


图 3-1 开发者模式

还可以通过在网页上右击，选择“检查”，或者按 Ctrl+Shift+I 组合键，如图 3-2 所示。



图 3-2 开发者模式

开发者工具的界面共有 9 个标签页，分别是：Elements、Console、Sources、Network、Performance、Memory、Application、Security 和 Audits。

Chrome 开发者工具以 Web 调试为主，如果用于爬虫分析，熟练掌握 Elements 和 Network 标签就能满足大部分的爬虫需求。其中，Network 是核心部分。

3.2 Elements 标签

在 Elements 标签页允许从浏览器的角度看页面，也就是说可以看到 Chrome 渲染页面所需要的 HTML、CSS 和 DOM（Document Object Model）对象。此外，还可以编辑内容更改页面显示效果，如图 3-3 所示。

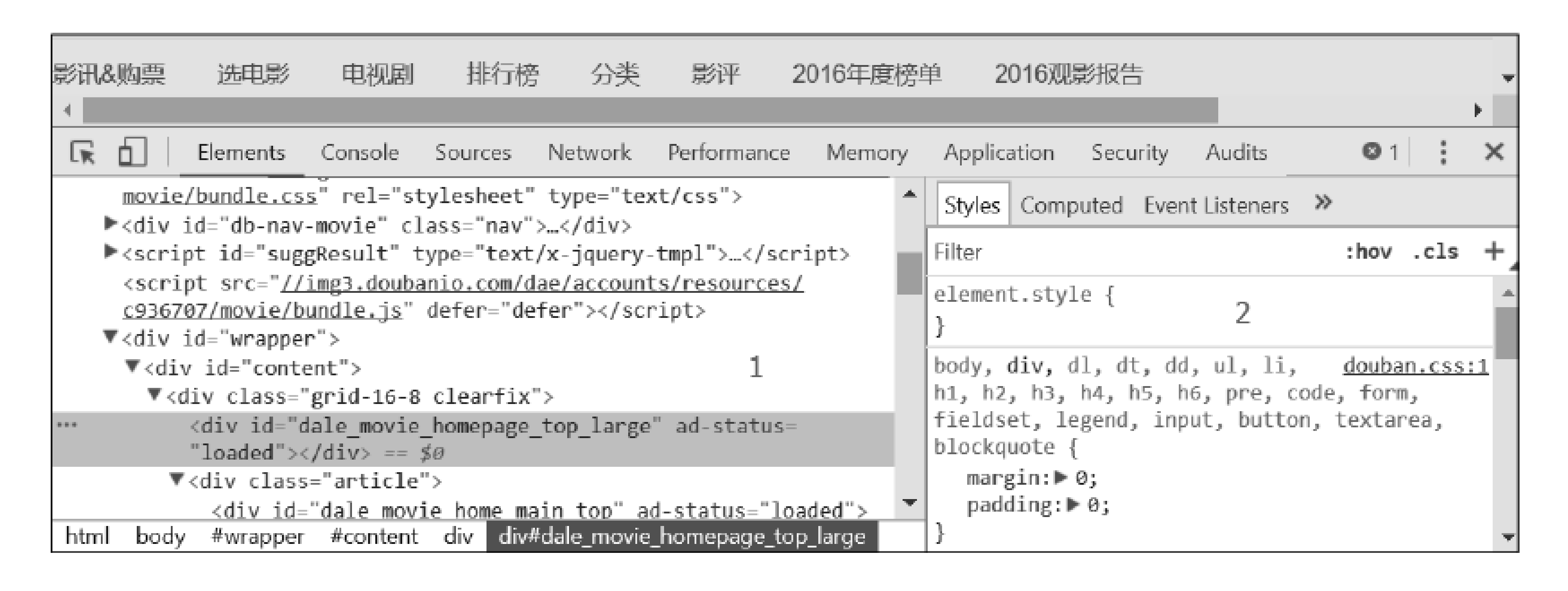


图 3-3 Elements 标签页

从图 3-3 可以看出，在 Elements 标签最左边的 按钮用于快速查找网页元素，单

击该按钮后，在网页上某一处单击，就会自动显示并选中该元素在 HTML 里的位置。

Elements 标签分成两部分，分别在图 3-3 中标为区域 1 和区域 2，两个区域相辅相成。区域 1 显示整个网页的 HTML 信息，单击选中某一行内容的时候，区域 2 的 Styles 标签会显示当前点击选中内容的 CSS 样式，并可对元素的 CSS 进行查看与编辑修改，Computed 显示当前选中的边距属性、边框属性，用图像显示一个整体效果，Event Listeners 是整个网页事件触发的 JavaScript，如图 3-4 所示。

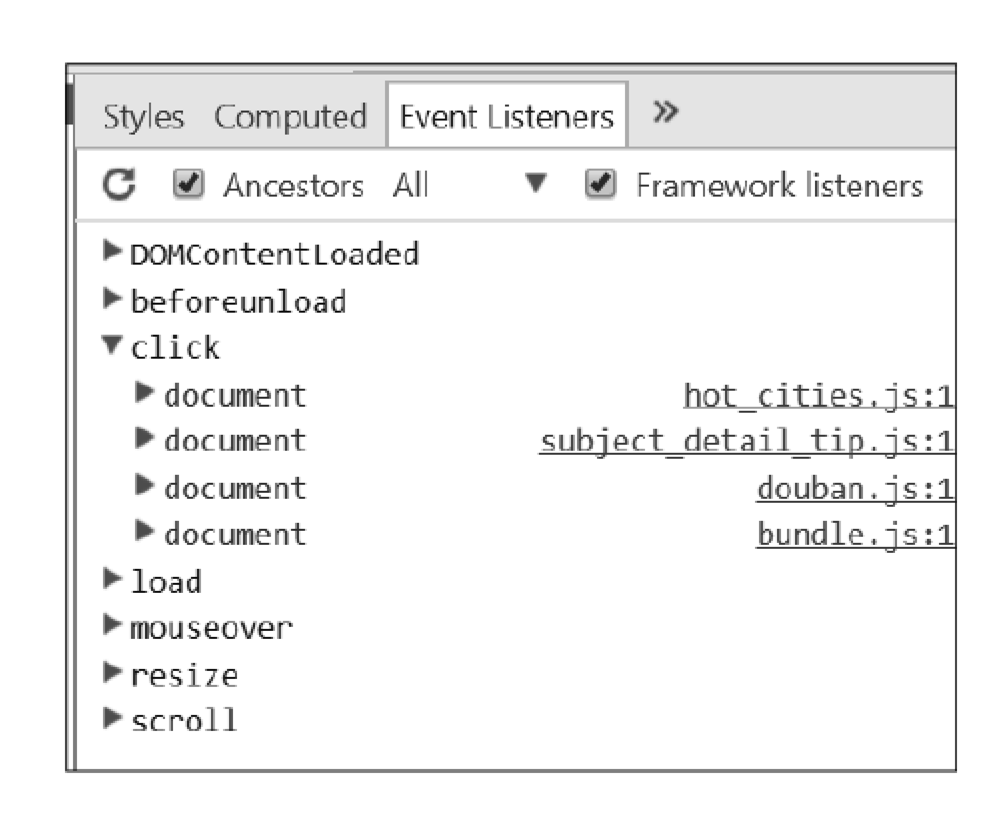


图 3-4 Event Listeners

通过单击 Event Listeners 下的某个 JavaScript 会自动跳转到 Sources 标签，显示当前 JavaScript 的源码，这个功能可快速找到 JavaScript 代码所在的位置，对分析 JavaScript 起到快速定位作用。

3.3 Network 标签

在 Network 标签页可以看到页面向服务器请求的信息、请求的大小以及加载请求花费的时间。从发起网页页面请求 Request 后分析 HTTP 请求得到各个请求信息（包括状态、类型、大小、所用时间、Request 和 Response 等）。Network 结构组成如图 3-5 所示。

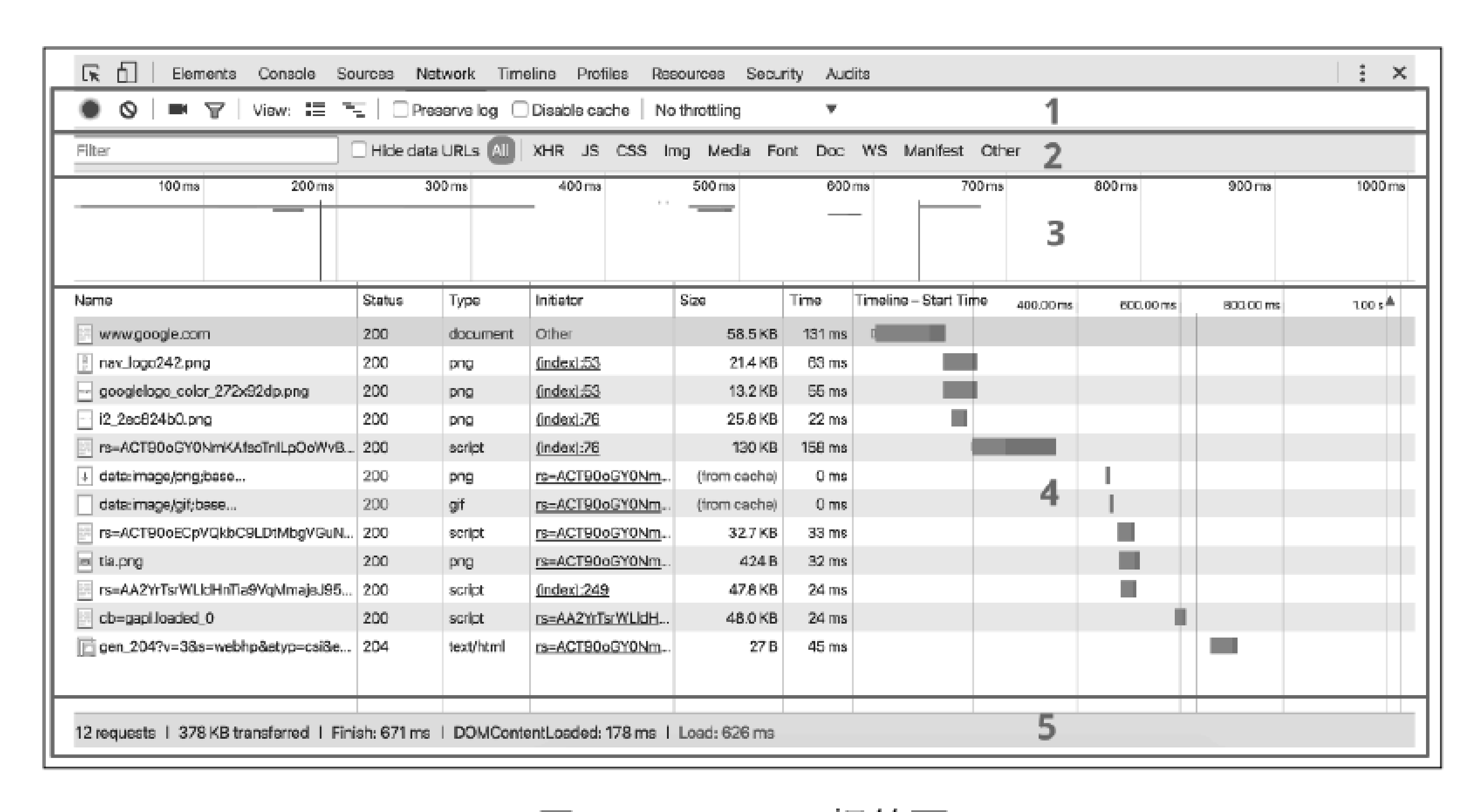


图 3-5 Network 标签页

Network 主要包括以下 5 个区域。

- Controls：控制 Network 的外观和功能。
- Filters：控制 Requests Table 具体显示哪些内容。
  - All：返回当前页面全部加载的信息，就是一个网页全部所需要的代码、图片等请求。
  - XHR：筛选 Ajax 的请求链接信息，前面讲过 Ajax 核心对象 XMLHTTPRequest，XHR 取于 XMLHTTPRequest 的缩写。
  - JS：主要筛选 JavaScript 文件。
  - CSS：主要是 CSS 样式内容。
  - Img：是网页加载的图片，爬取图片的 URL 都可以在这里找到。
  - Media：是网页加载的媒体文件，如 MP3、RMVB 等音频视频文件资源。
  - Doc：是 HTML 文件，主要用于响应当前 URL 的网页内容。
- Overview：显示获取到请求的时间轴信息，主要是对每个请求信息在服务器的响应时间进行记录。这个主要是为网站开发优化方面提供数据参考，这里不做详细介绍。
- Requests Table：按前后顺序显示所有捕捉的请求信息，单击请求信息可以查看该详细信息。
- Summary：显示总的请求数、数据传输量、加载时间信息。

5 个区域中，Requests Table 是核心部分，主要作用是记录每个请求信息。但每次网站出现刷新时，请求列表都会清空并记录最新的请求信息，如用户登录后发生 304 跳转，就会清空跳转之前的请求信息并捕捉跳转后的请求信息。

对于每条请求信息，可以单击查看该请求的详细信息，如图 3-6 所示。

每条请求信息划分为以下 5 个标签。

- Headers：该请求的 HTTP 头信息。
- Preview：根据所选择的请求类型（JSON、图片、文本）显示相应的预览。
- Response：显示 HTTP 的 Response 信息。

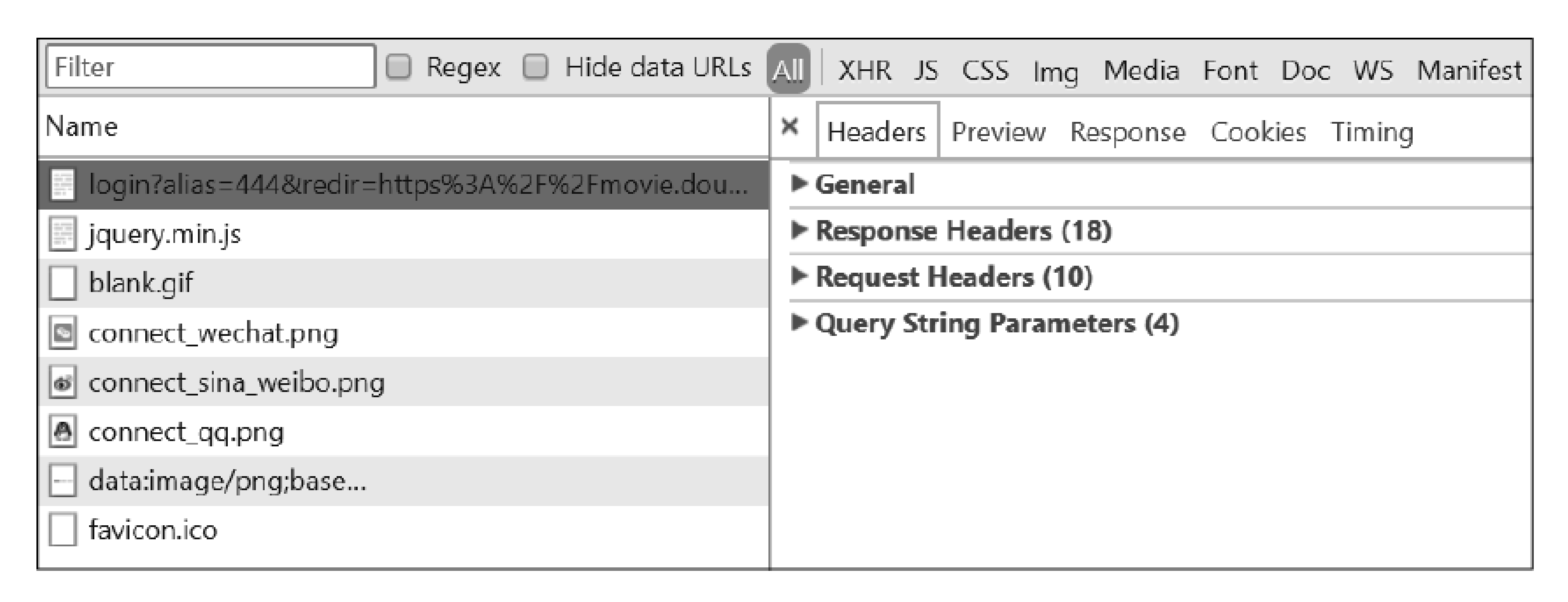


图 3-6 请求信息

- Cookies：显示 HTTP 的 Request 和 Response 过程中的 Cookies 信息。
- Timing：显示请求在整个生命周期中各部分花费的时间。

常用的标签有 Headers、Preview 和 Response。Headers 用于获取请求链接、请求头和请求参数；Preview 和 Response 用于显示服务器返回的响应内容。

Headers 标签划分为以下 4 部分。

- General：记录请求链接、请求方式和请求状态码。
- Response Headers：服务器端的响应头。其参数说明如下。
  - Cache-Control：指定缓存机制，优先级大于 Last-Modified。
  - Connection：包含很多标签列表，其中最常见的是 Keep-Alive 和 Close，分别用于向服务器请求保持 TCP 连接和断开 TCP 连接。
  - Content-Encoding：服务器通过这个头告诉浏览器数据的压缩格式。
  - Content-Length：服务器通过这个头告诉浏览器回送数据的长度。
  - Content-Type：服务器通过这个头告诉浏览器回送数据的类型。
  - Date：当前时间值。
  - Keep-Alive：在 Connection 为 Keep-Alive 时，该字段才有用，用来说明服务器估计保留连接的时间和允许后续几个请求复用这个保持着的连接。
  - Server：服务器通过这个头告诉浏览器服务器的类型。
  - Vary：明确告知缓存服务器按照 Accept-Encoding 字段的内容分别缓存不同的版本。

- Request Headers：用户的请求头。其参数说明如下。
  - Accept：告诉服务器客户端支持的数据类型。
  - Accept-Encoding：告诉服务器客户端支持的数据压缩格式。
  - Accept-Charset：可接受的内容编码 UTF-8。
  - Cache-Control：缓存控制，服务器控制浏览器要不要缓存数据。
  - Connection：处理完这次请求后，是断开连接还是保持连接。
  - Cookie：客户可通过 Cookie 向服务器发送数据，让服务器识别不同的客户端。
  - Host：访问的主机名。
  - Referer：包含一个 URL，用户从该 URL 代表的页面出发访问当前请求的页面，当浏览器向 Web 服务器发送请求的时候，一般会带上 Referer，告诉服务器请求是从哪个页面 URL 过来的，服务器借此可以获得一些信息用于处理。
  - User-Agent：中文名为用户代理，简称 UA，是一个特殊字符串头，使得服务器能够识别客户使用的操作系统及版本、CPU 类型、浏览器及版本、浏览器渲染引擎、浏览器语言、浏览器插件等。
- Query String Parameters：请求参数。主要是将参数按照一定的形式（GET 和 POST）传递给服务器，服务器通过接收其参数进行相应的响应，这是客户端和服务端进行数据交互的主要方式之一。

Headers 标签的内容看起来很多，但在实际使用过程中，爬虫开发人员只需关心请求链接、请求方式、请求头和请求参数的内容即可。

而 Preview 和 Response 是服务器返回的结果，两者之间对不同类型的响应结果有不同的显示方式：

（1）如果返回的结果是图片，那么 Preview 表示可显示图片内容，Response 表示无法显示。

（2）如果返回的是 HTML 或 JSON，那么两者皆能显示，但在格式上可能会存在细微的差异。

3.4 分析 QQ 音乐

现在以QQ音乐某一歌手页面的分析为例（y.qq.com/n/yqq/singer/0025NhlN2yWrP4.html）讲述如何使用 Chrome 开发者工具分析网站，如图 3-7 所示。

图 3-7 歌手信息

从图 3-7 中可以看到，在 Network 标签下捕捉到很多请求信息，请求类型有document、png、font 和 script 等，分别对应 HTML 文件、图片、字体格式和 JS 脚本。

单击“Filters”下的 Doc 标签（Doc 是当前网页的 HTML 文件），发现有两个请求信息，分别是“0025NhlN2yWrP4.html”和“xhr_proxy_utf8.html”。从请求的命名可以看出，第一个请求与网站的 URL 是一致的。再查看“0025NhlN2yWrP4.html”的响应内容（Preview 标签），可以使用“Ctrl+F”快速查找歌曲信息，如图 3-8 所示。



图 3-8 快速查找歌曲信息

在 Doc 中虽能找到歌曲名、专辑和时长，但无法找到更多的歌曲信息。歌曲信息有可能是由其他方式生成的，网站数据生成只有前端（Ajax 或 JSONP）和后端（服务器）

两种方式。从图 3-8 返回的结果来看，数据不可能是从后端生成的，那么就可能是由前端加载生成的。



JSONP（JSON With Padding）是 JSON 的一种“使用模式”，可用于解决主流浏览器的跨域数据访问问题。

前端加载的数据有可能记录在 Chrome 开发者工具的“XHR”或“JS”中，分别查看两个标签里面的请求信息，最终发现歌曲信息存放在 JS 下的某个请求中，如图 3-9 和图 3-10 所示。

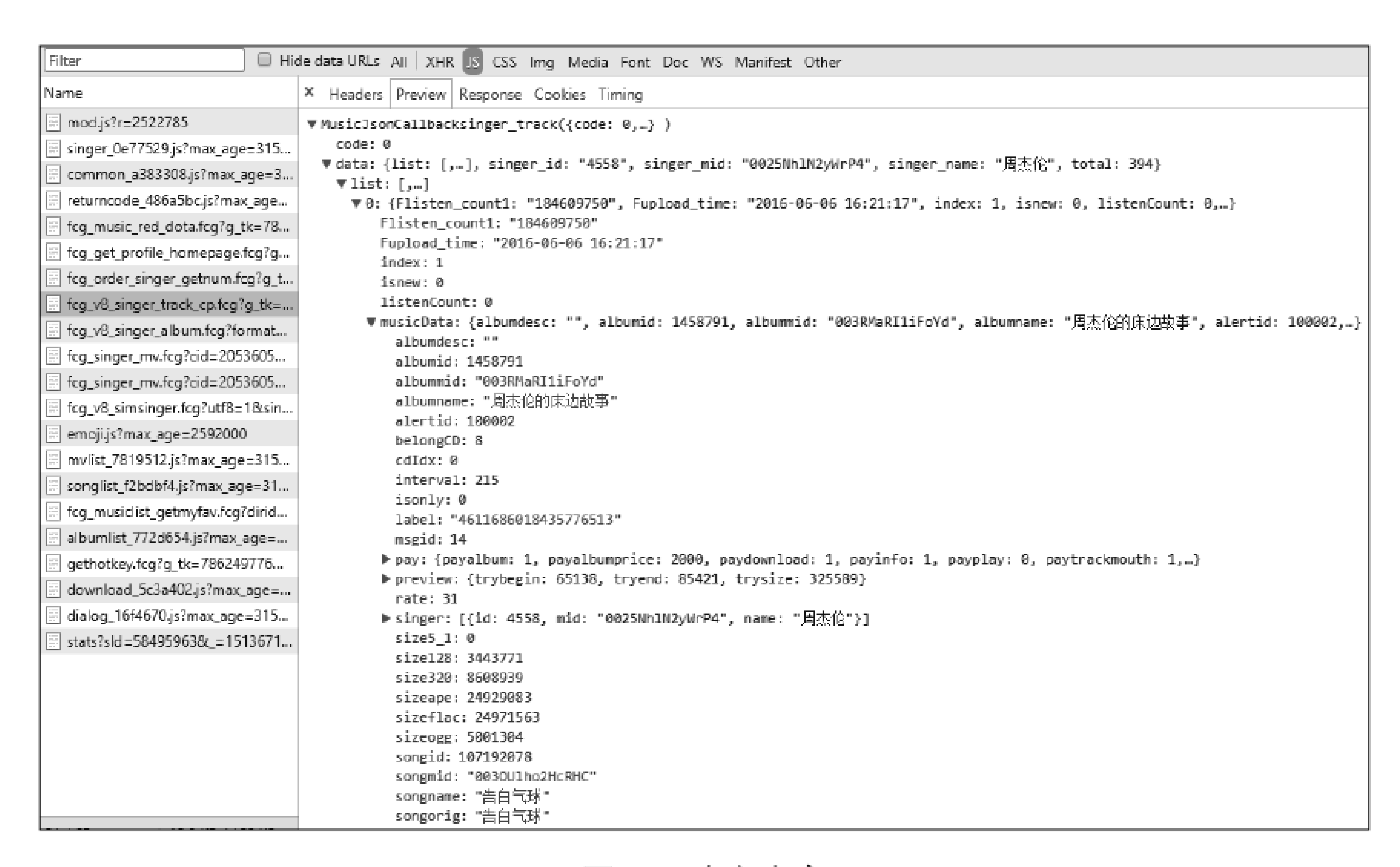


图 3-9 响应内容

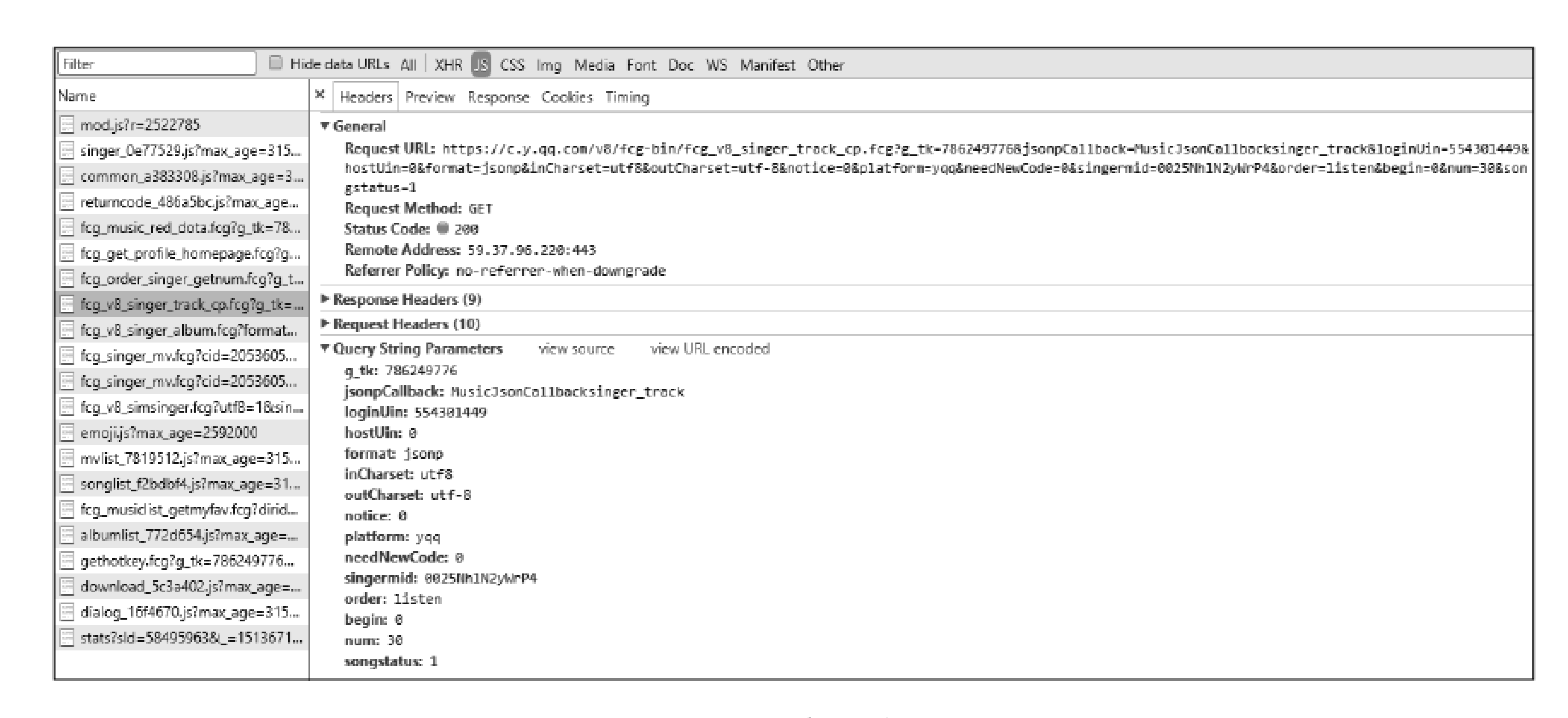


图 3-10 请求信息

从图 3-9 和图 3-10 分析得知，请求方式是 GET，Query String Parameters 是记录该请求的参数。因为请求方式是 GET，所以请求参数也可以在请求链接上找到。

再看请求参数，大部分请求参数是可以明确知道的，唯独参数 singermid 无法确定。从参数的命名来看，这应该是歌手的 ID 信息，也是网站用于标记歌手唯一的属性，所以要获取歌手的 singermid 可能需要从其他请求上获取。

根据上述例子，可简单总结出分析网站的步骤如下：

步骤01 找出数据来源，大部分数据来源于 Doc、XHR 和 JS 标签。

步骤02 找到数据所在的请求，分析其请求链接、请求方式和请求参数。

步骤03 查找并确定请求参数来源。有时候某些请求参数是通过另外的请求生成的，比如请求 A 的参数 id 是通过请求 B 所生成的，那么要获取请求 A 的数据，就要先获取请求 B 的数据作为 A 的请求参数。

3.5 本章小结

Chrome 开发者工具的主要作用是进行 Web 开发调试，对于爬虫开发人员来说，应该熟练掌握 Elements、Console 和 Network。其中 Network 是核心部分，百分之九十的网站分析都在 Network 上完成，读者对 Network 上的各个功能和作用要理解掌握，并懂得如何使用 Chrome 分析网站的请求信息。

一般分析网站最主要的是找到数据的来源，确定数据来源就能确定数据生成的具体方法。总结归纳分析网站的步骤如下：

（1）找出数据来源，大部分数据来源于 Doc、XHR 和 JS 标签。

（2）找到数据所在的请求，分析其请求链接、请求方式和请求参数。

（3）查找并确定请求参数来源。有时候某些请求参数是通过另外的请求生成的，比如请求 A 的参数 id 是通过请求 B 所生成的，那么要获取请求 A 的数据，就要先获取请求 B 的数据作为 A 的请求参数。

上述分析步骤适用于大部分网站分析，但每个网站都有自身设计的特点，不能一概而论。此方法更多的是起到指导性作用，遇到具体的问题还是要具体分析。

第 4 章

Fiddler 抓包工具

4.1 Fiddler 介绍

Fiddler 是一款非常流行并且实用的 HTTP 抓包工具，原理是在电脑上开启一个 HTTP 代理服务器，然后转发所有的 HTTP 请求和响应。因此，比一般的浏览器自带的抓包工具（开发者工具）要好用得多。不仅如此，还可以支持请求重放一些高级功能，也可以支持对手机应用进行 HTTP 抓包。

Fiddler 是用 C# 开发的工具，包含一个简单却功能强大的基于 JScript .NET 事件的脚本子系统，灵活性非常棒，可以支持众多的 HTTP 调试任务，并且能够使用 .net 框架语言进行扩展。

此外，还支持断点调试技术，当请求或响应属性能够跟目标的标准相匹配时，

Fiddler 就能够暂停 HTTP 通信，并且允许修改请求和响应。这种功能对于安全测试非常有用，当然也可以用来做一般的功能测试。

4.2 Fiddler 安装配置

Fiddler 在 Windows 下可直接使用 exe 安装包安装，安装包可在官方网站下载（https://www.telerik.com/download/fiddler）。

完成安装后，在安装目录下双击打开应用程序“Fiddler.exe”，可看到 Fiddler 用户界面，如图 4-1 所示。

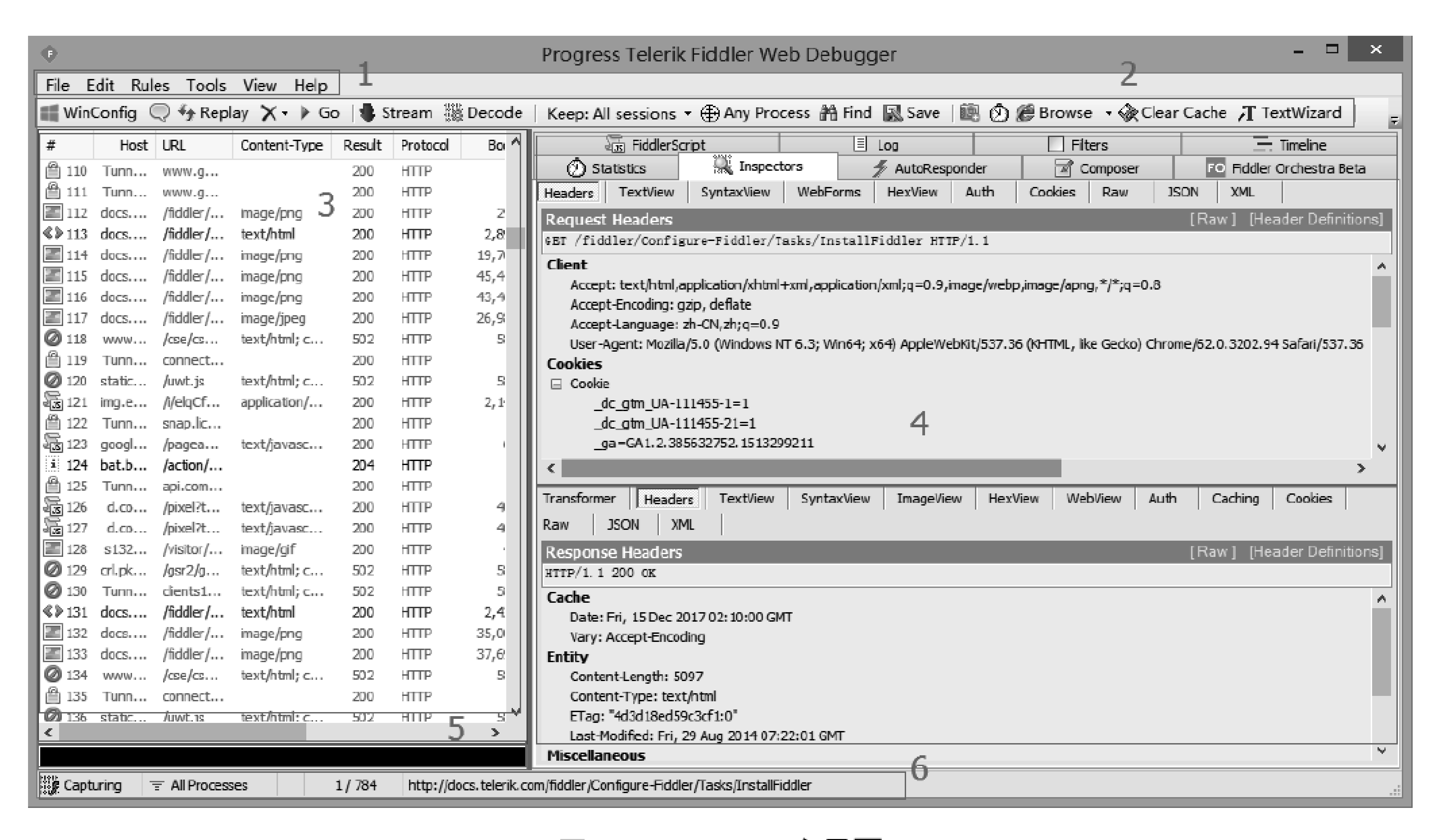


图 4-1　Fiddler 用户界面

Fiddler 用户界面主要包括下面 6 个部分：

（1）图中标注 1 为 Main Menu（主菜单），作用于整个 Fiddler 相关配置。

（2）图中标注 2 为 Toolbar（工具栏），主要对 Web Session 操作处理。

（3）图中标注 3 为 Web Session（列表），显示已抓取的 HTTP 请求信息。

（4）图中标注 4 为 View（选项视图），显示每条 HTTP 的详细信息。

（5）图中标注 5 为 Quickexec（命令行），通过特定的条件快速找到符合条件的 HTTP 请求。

（6）图中标注 6 为 Status bar（状态栏），显示当前状态信息。

打开 Fiddler 之后，由于 HTTPS 协议的特殊性，还需要配置 Fiddler。了解 Fiddler 抓取 HTTPS 协议的原理才能更好地理解如何对 Fiddler 进行配置，原理如图 4-2 所示。

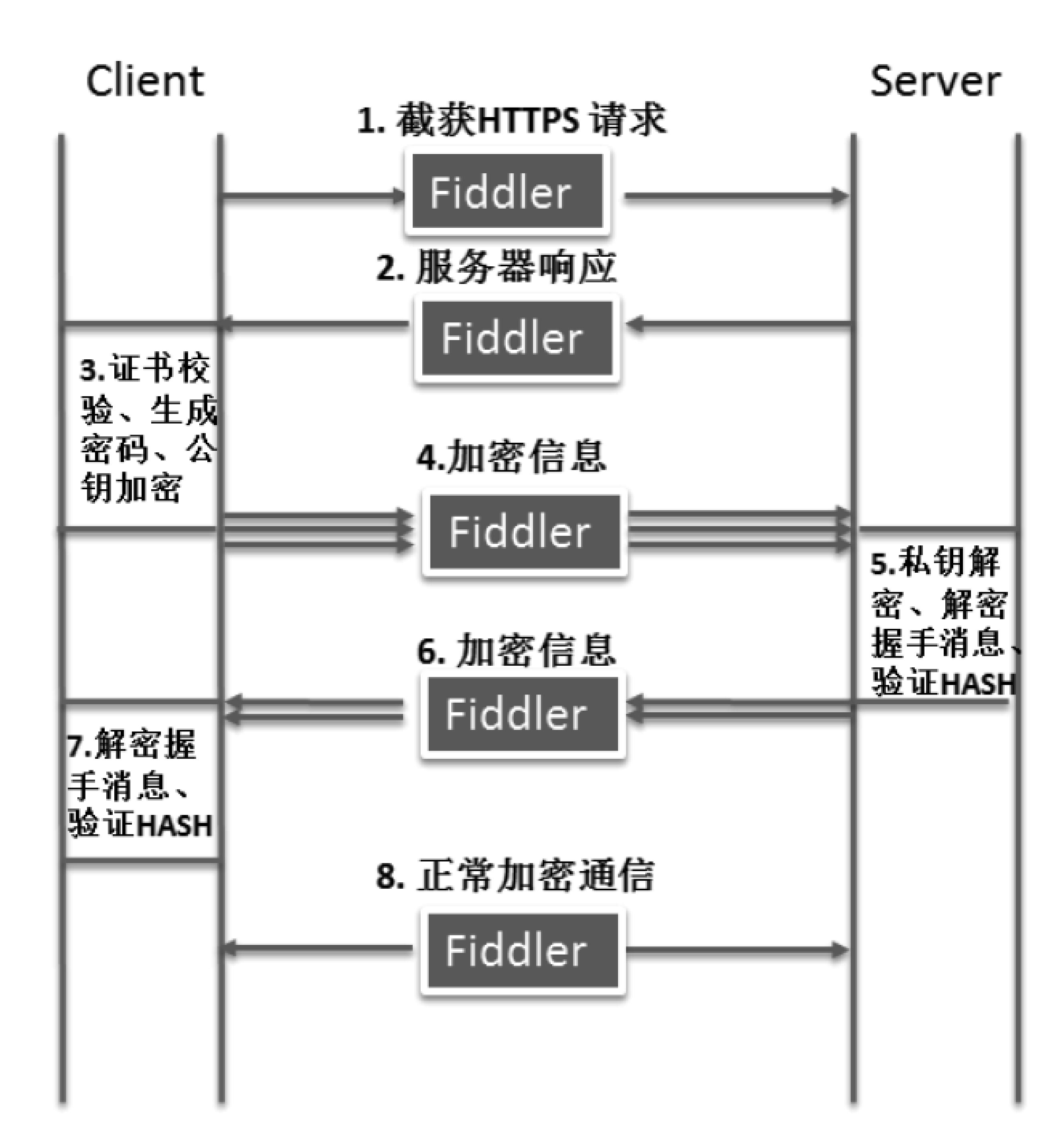


图 4-2 Fiddler 抓取 HTTPS 的原理

Fiddler 抓取 HTTPS 协议充当角色：

（1）服务器→客户端：Fiddler 接收到服务器发送的密文，用对称密钥解开，获得服务器发送的明文。再次加密，发送给客户端。

（2）客户端→服务器端：客户端用对称密钥加密，被 Fiddler 截获后，解密获得明文。再次加密，发送给服务器端。由于 Fiddler 一直拥有通信用对称密钥 enc_key，因此在整个 HTTPS 通信过程中信息对其透明。

配置 Fiddler，使其能够抓取 HTTPS 请求信息，方法如下：

步骤 01 对 Fiddler 进行设置：打开 Main Menu → Tools → Fiddler Options → HTTPS。

步骤 02 勾选 HTTPS 里的选项，然后单击 Actions → Trust Root Certificate，完成证书验证，如图 4-3 所示。

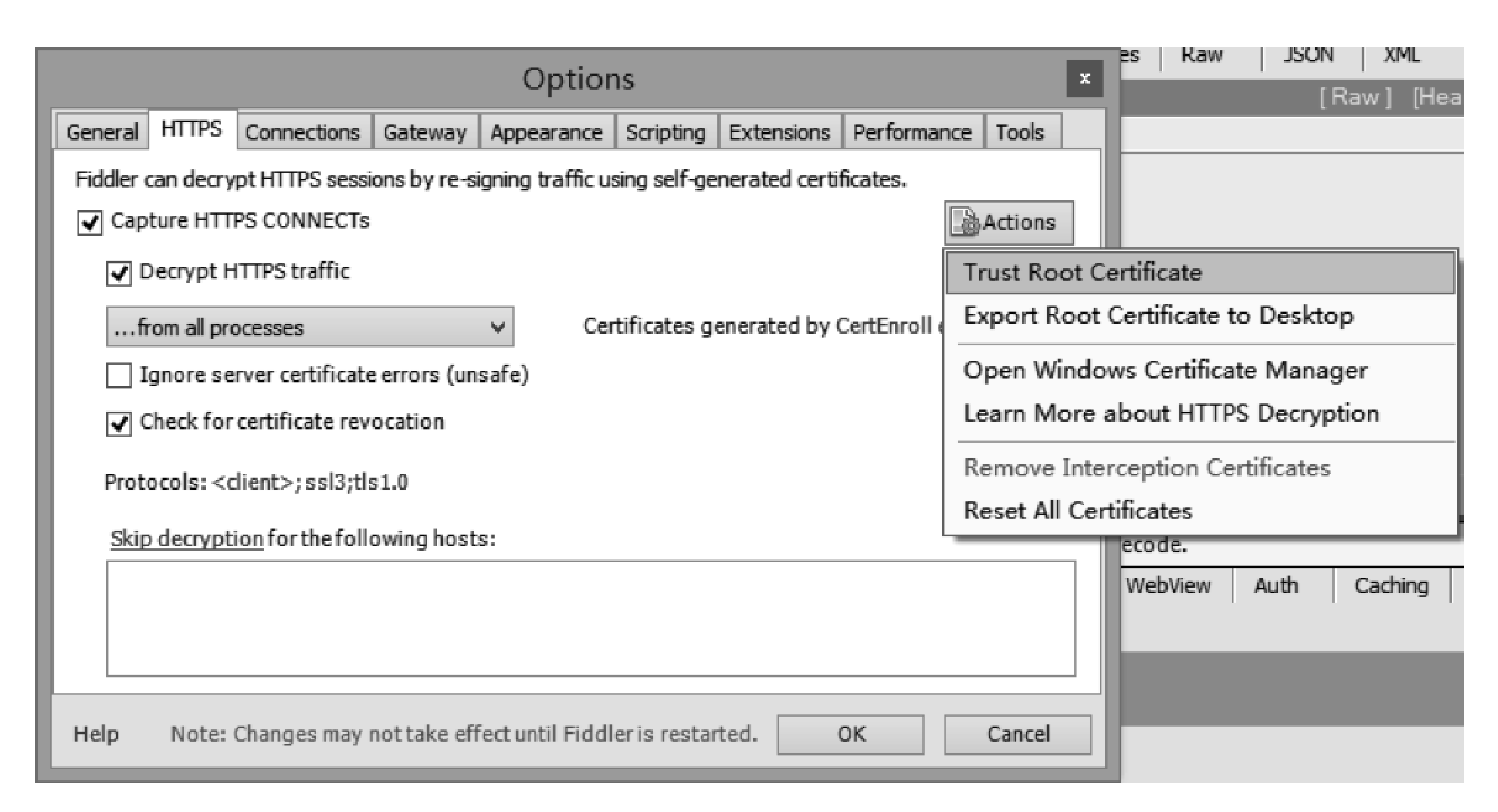


图 4-3 Fiddler 配置 HTTPS

步骤 03 完成配置，重启浏览器，Fiddler 就能正常抓取 HTTPS 请求信息。

完成上述安装和配置，Fiddler 就能抓取浏览器的请求信息。除此之外，Fiddler 还能抓取手机上的请求信息，具体使用方法在后续章节会详细讲述。

4.3 Fiddler 抓取手机应用

Fiddler 可通过同一无线网络实现对手机应用的抓包，手机抓包原理和电脑抓包原理相同，手机抓包主要通过远程连接实现手机和 Fiddler 通信。

实现 Fiddler 抓取手机应用的步骤如下：

步骤 01 配置 Fiddler 远程连接模式。打开 Main Menu → Tools → Fiddler Options → Connections，勾选 Allow remote computers to connect 复选框，如图 4-4 所示。

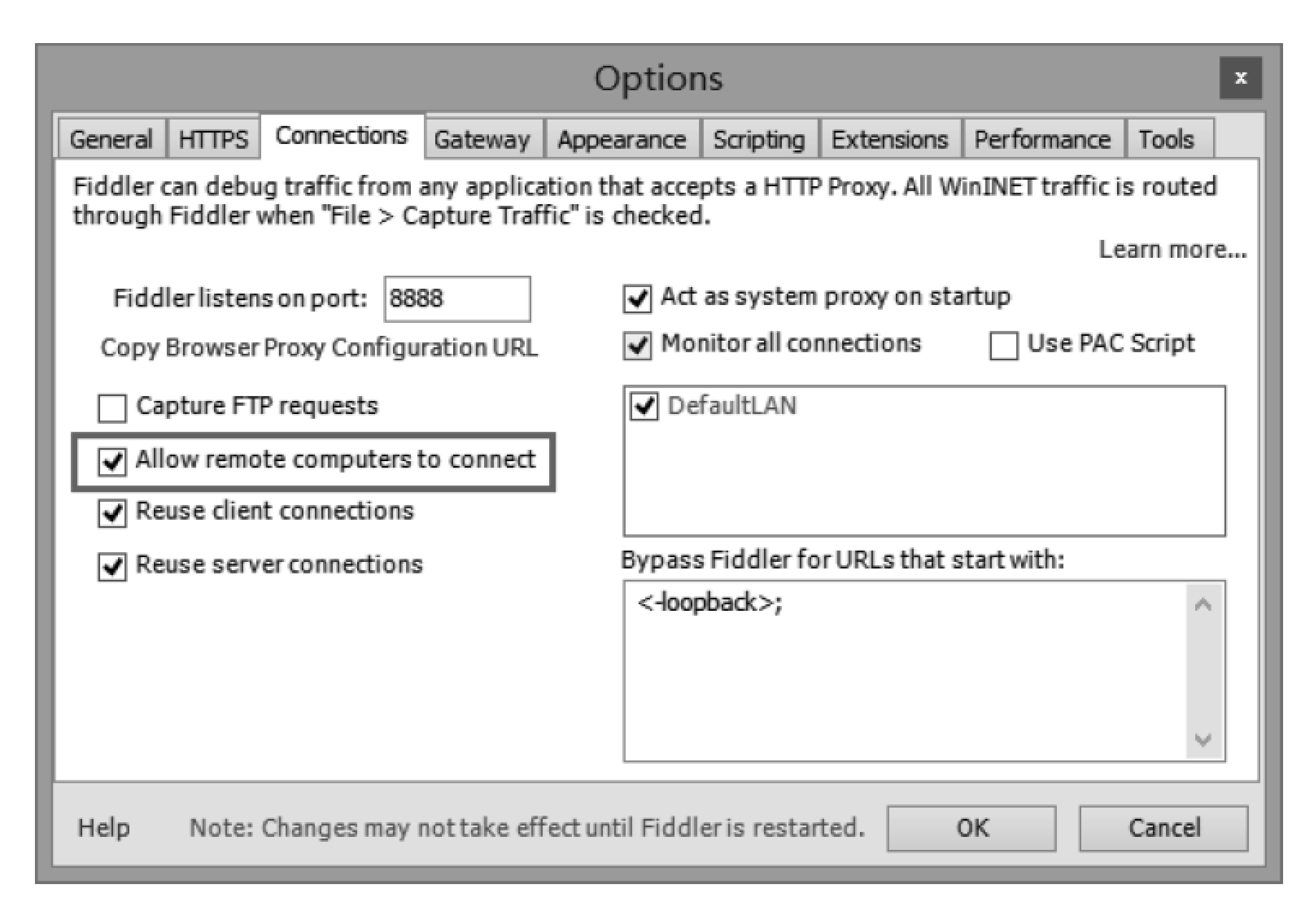


图 4-4 Fiddler 配置远程连接

步骤 02 在手机端进行参数配置（以安卓手机为例）。确保手机和电脑在同一个网络，查询电脑 IP 地址，可在 CMD 下输入 ipconfig 查询（电脑 IP 为 10.168.1.240），从图 4-4 得知 Fiddler 端口为 8888（一般默认为 8888，也可自行设置）。

步骤 03 在手机浏览器中输入电脑 IP 地址和 Fiddler 端口（输入“10.168.1.240: 8888”），点击确认后跳转到证书下载页面。点击下载 FiddlerRoot certificate，如图 4-5 所示。

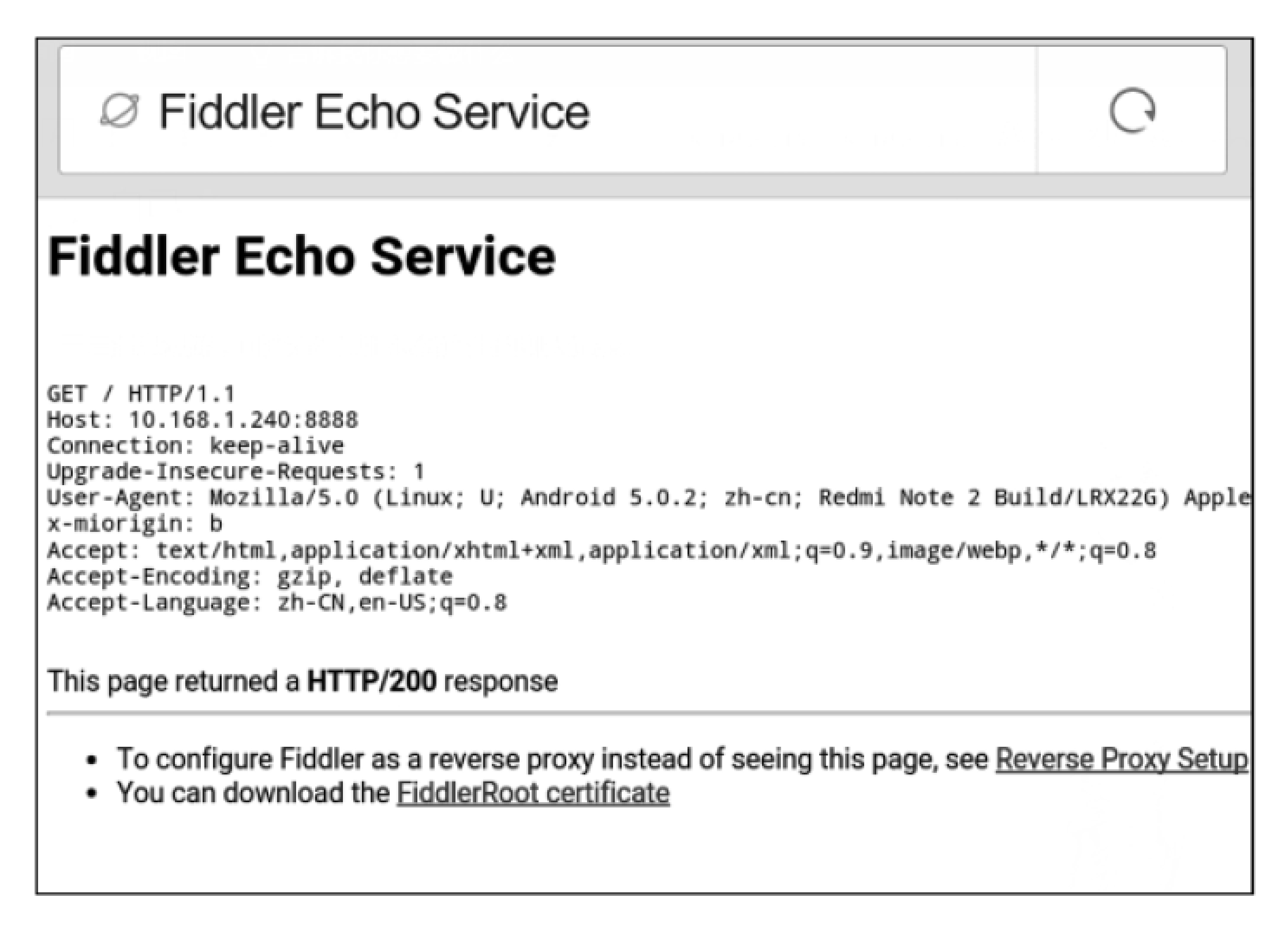


图 4-5 下载 FiddlerRoot certificate

步骤 04　证书文件以 cer 为后缀名，由于不同的手机型号安装证书的方式不一致，因此这里不做详细讲述。完成证书安装后，进入手机当前连接 Wi-Fi 详情，设置代理 IP：主机名为电脑 IP 地址，端口为 Fiddler 配置的端口，如图 4-6 所示。



图 4-6 配置手机代理

步骤 05　完成上述配置，可操作手机应用，在操作过程中产生的 HTTP 请求都会被电脑上的 Fiddler 抓取，如图 4-7 所示。

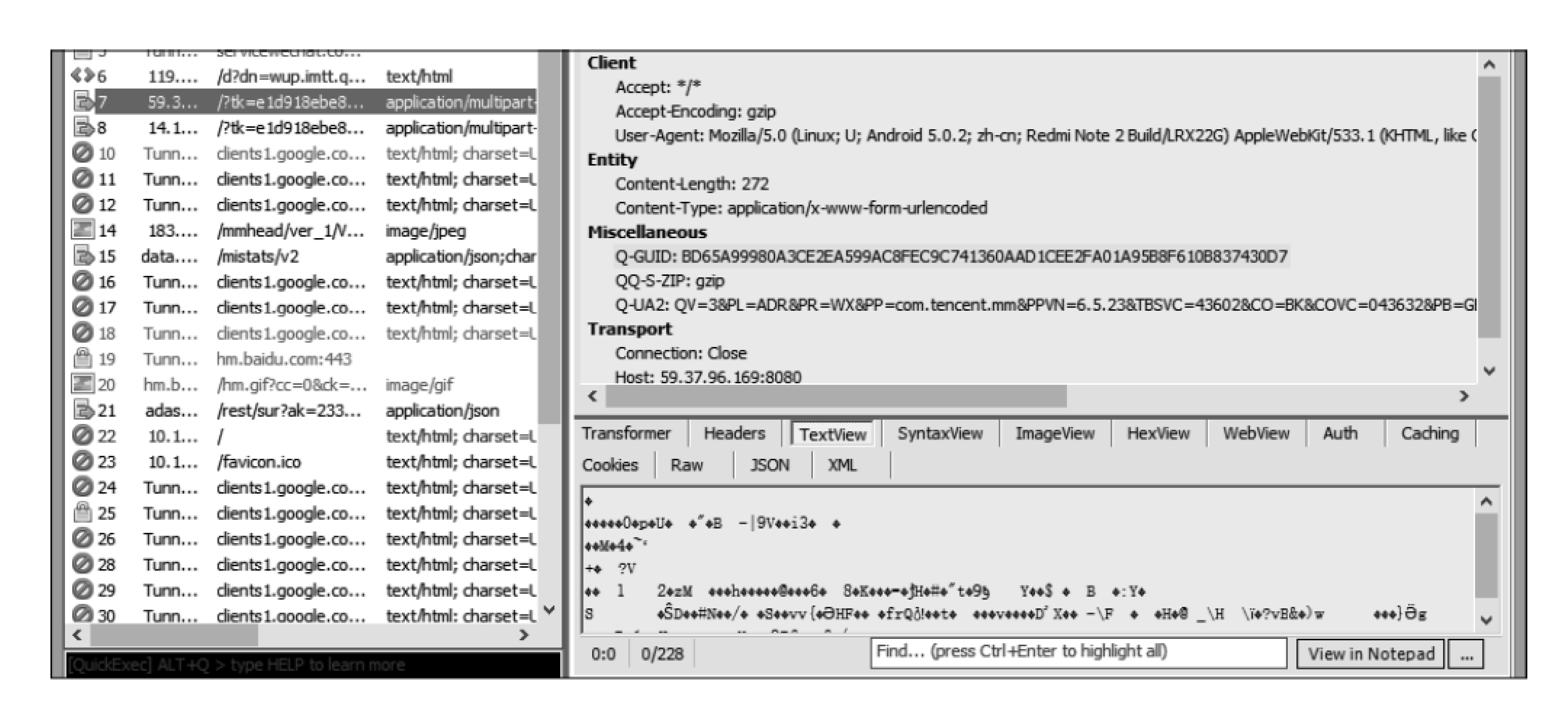


图 4-7 Fiddler 抓取手机 HTTP 请求信息

从图4-7中看到，User-Agent请求头是由安卓系统发出的。同样地，若抓取iOS系统，也是按照上述方式配置即可。

如果停止电脑对手机的网络监控，可以回到步骤 4，将 Wi-Fi 的代理设置去掉即可。若要删除 Fiddler 证书，可在设置→系统安全→信任的凭据下找到“DO_NOT_TRUST”证书，将其删除即可。

4.4 Toolbar 工具栏

现在能使用 Fiddler 抓取 HTTP 请求信息，但如何使用 Fiddler 对请求进行分析，从而获取我们所需的信息呢？要熟练掌握 Fiddler，首先要掌握其每个功能的作用。

从 4.2 节知道，Fiddler 用户界面由 6 部分组成，最为常用的是 Toolbar 工具栏、Web Session 列表、View 选项视图和 Quickexec 命令行。

Toolbar 工具栏主要提供常见的命令和设置的快捷方式，其各个按钮的功能说明如表 4-1 所示。

表4-1 Toolbar工具栏功能说明

快捷键	含　义
	单击该按钮可以为所有选定的Session添加comment
Replay	向服务器重新发送该请求
X	从Web Session中删除已经捕捉的Session
Go	恢复执行在request或response断点处暂停的所有Session
Stream	打开Stream模式，取消所有没有设置中断的缓存
Decode	打开Decode模式，对请求和响应的HTTP内容和传输编码进行解码
Keep: All sessions	选择Web Session列表中保存Session的数量
Any Process pick target...	单击上面的Any Process图标并将其移动到指定浏览器页面后，该log会单独记录这个页面的通信情况
Find	打开Find Session窗口，可快速查找某条Session
Save	把所有的Session保存到saz文件中
(2...)	把当前桌面的屏幕截图以JPEG格式添加到Web Session列表
	简单的计时功能

快捷键	含　义
Browse	如果选中某个Session，就会在IE中打开该URL；如果没有选中Session，就在IE中打开about:blank
Clear Cache	清空WinINET的缓存文件
TextWizard	打开文本编码/解码小工具
Tearoff	新建一个包含所有View的窗口
MSDN Search...	在MSDN的Web Session区域进行搜索
?	打开Fiddler的帮助窗口
X	删除工具栏，如果要恢复工具栏，可单击View→Show Toolbar

4.5 Web Session 列表

Web Session 主要以表格形式展现，需掌握表字段代表的含义、表格信息的含义以及快捷键的使用。

表字段（表头）信息的含义如表 4-2 所示。

表4-2 Web Session表头信息及说明

表　头	含义与作用
#	对已捕捉的Session生成对应的ID号
Host	接受请求的主机名和端口
URL	请求URL的路径
Content-Type	Session的内容类型
Result	响应的状态码
Protocol	网络协议类型（HTTP、HTTPS、FTP）
Body	包含的字节数
Caching	响应头中Expires和Cache-Control的值
Process	数据流在本地系统的进程
Comments	通过工具栏Comment按钮设置注释信息
Custom	FiddlerScript所设置的Ui-CustomColumn标志位的值

观察列表每条请求信息可发现，每条数据都有不同的颜色，其颜色含义如表 4-3 所示。

表4-3 Web Session请求信息的颜色含义

颜　色	含　义
红色	表示HTTP状态错误
黄色	表示HTTP状态需用户认证
灰色	表示数据流类型CONNECT或表示响应内容是图像
紫色	表示响应内容是CSS文件
蓝色	表示响应内容是HTML
绿色	表示响应内容是Script文件

除了颜色之外，每条请求信息都带有一个图标，图标含义如表 4-4 所示 。

表4-4 Web Session请求信息的图标含义

图　标	含　义
	正在向服务器发送请求
	正在向服务器获取响应
	请求停止于断点处，允许对它进行修改
	响应停止于断点处，允许对它进行修改
	请求使用的是Head或Options方法，客户端无须下载内容即可获取目标URL和服务器信息
	请求使用POST方法向服务器发送数据
	响应内容是HTML
	响应内容是图像文件
	响应内容是脚本文件
	响应内容是CSS文件
	响应内容是XML格式文件
	响应内容是JSON格式文件
	响应内容是音频文件
	响应内容是视频文件
	响应内容是SilverLight程序
	响应内容是Flash应用程序
	响应内容是字体文件
	响应成功
	Content-Type内容是Text/HTML
	响应是HTTP / 300、301、302、303、307重定向
	要求对客户端进行认证
	服务器返回错误的标识

（续表）

图　标	含　义
	Session被客户端或服务器中止
	响应内容使用缓存版本

通过对表字段、请求信息的颜色和请求信息的图标的了解，可知道 Fiddler 对每一种请求类型都进行了详细的划分。除此之外，用户还可以对每个请求进行操作，移动鼠标对某一条请求右击会出现操作菜单，如图 4-8 所示。

用户可通过操作菜单对请求信息进行更改，也可以直接使用快捷键实现，快捷键表 4-5 所示。（图中颜色较深的是常用部分。）

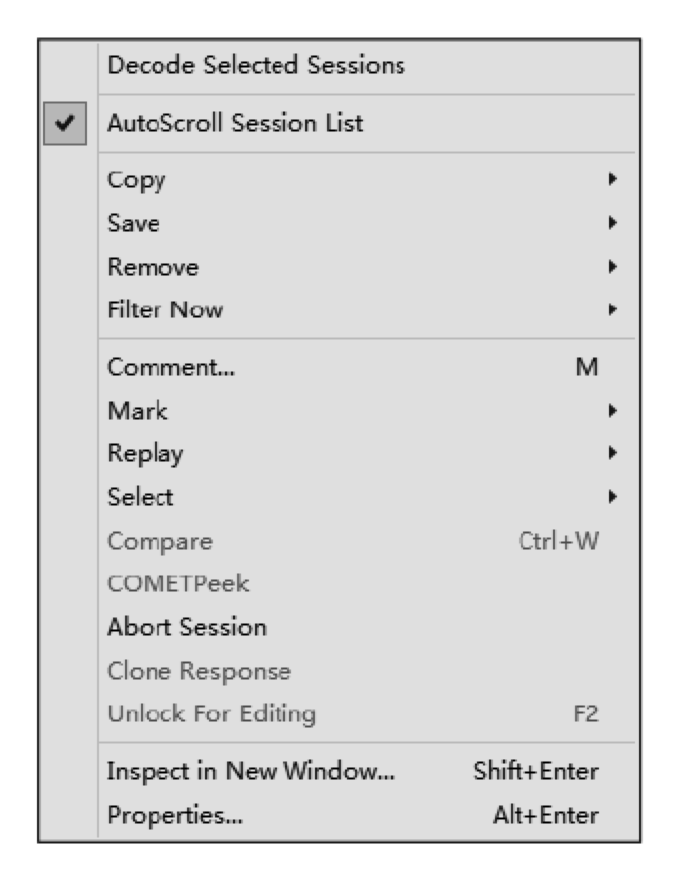


图 4-8　Web Session 操作菜单

表4-5　Web Session快捷键

快捷键	含　义
SpaceBar	在视图中激活并显示当前的Session
Ctrl + A	选中所有的Session
ESC	取消选择所有的Session
Ctrl + I	反向选中，取消选中的Session，选中之前未选中的Session
Ctrl + X	删除所有Session
Delete	删除选中的Session
Shift + Delete	删除所有未选中的Session
R	重新执行当前请求
Shift + R	多次执行当前的请求（在提示框输入执行次数）
U	无条件重新执行当前的请求
Shift +U	无条件多次执行当前的请求（在提示框输入执行次数）
P	选中触发该请求的父请求
C	选中该响应触发的所有子请求
D	选中与当前Session重复的请求
Alt + Enter	查看当前Session的属性
Shift + Enter	在新的Fiddler窗口中启动该Session的Inspectors
Ctrl + 1/2/3/4/5/6	选中的Session分别用粗体的红色/蓝色/金色/绿色/橙色/紫色表示
M	为选中的Session添加描述

4.6 View 选项视图

View 主要以选项视图方式实现，将一个请求信息按功能划分在不同选项卡中。如表 4-6 所示是常用的选项视图。

表4-6 View常用选项视图

常用选项视图	含　义
Statistics	统计选项卡，统计资源的消耗时间和数据长度等信息
Inspectors	检查选项卡，共分为两部分：请求信息和响应信息
AutoResponder	自动响应选项卡，将请求重定向到本地文件，实现人工干预HTTP请求
Composer	构建选项卡，用于创建HTTP Request，并发送服务器实现模拟请求

使用 Fiddler 做爬虫开发必须掌握 Inspectors、AutoResponder 和 Composer。Inspectors 主要是对请求和响应信息进行分析和获取请求参数，如图 4-9 所示。

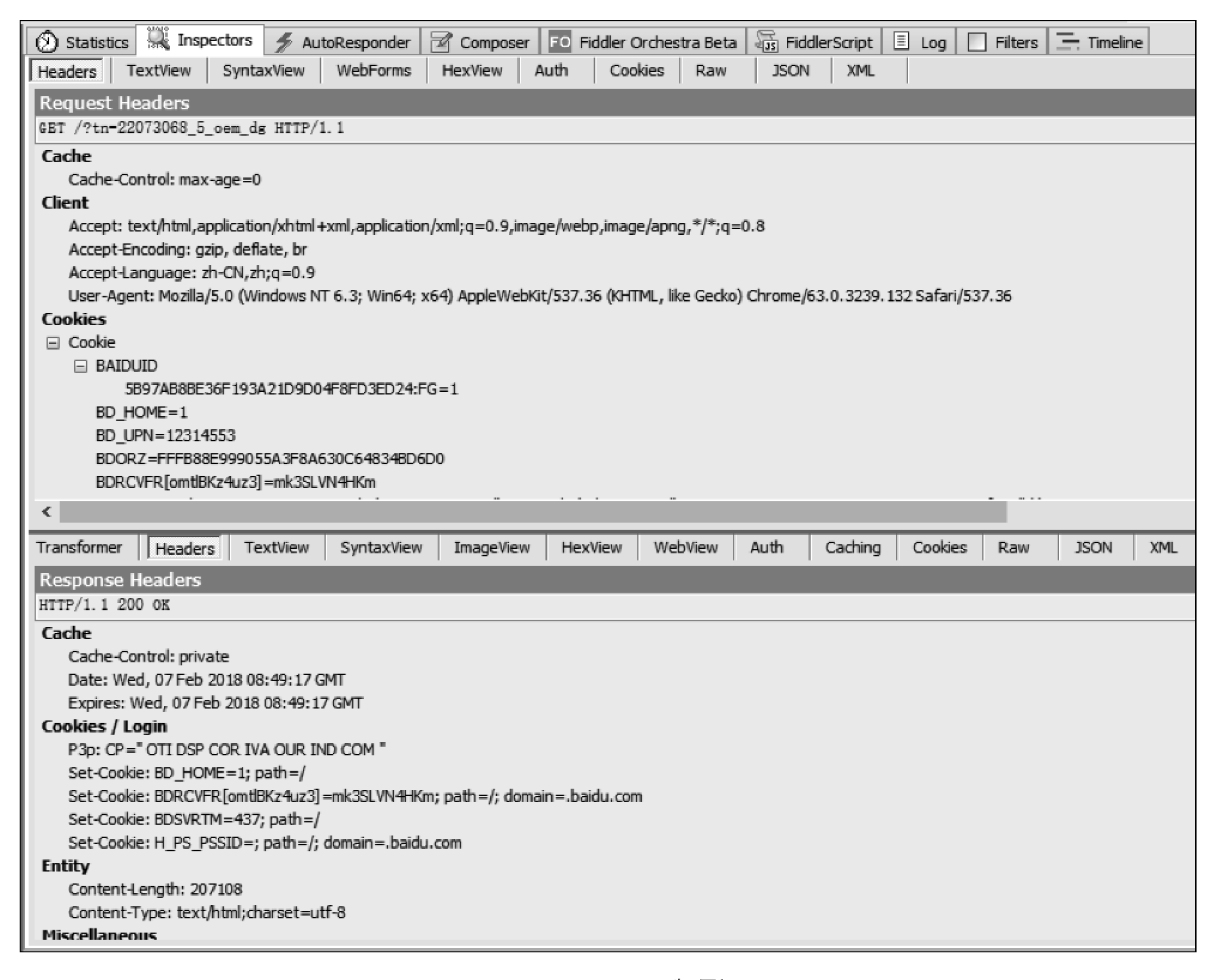


图 4-9 Inspectors 选项视图

从图 4-9 知道，Inspectors 上下划分为两个功能区。上面的功能区显示用户发送的请求信息，下面的功能区显示服务器响应的内容，每个功能区里又划分了多个选项视图。其中，Headers 选项视图显示请求头和响应头，WebForm 选项视图显示发送请求的请求参数，xxxView 选项卡显示各种类型的数据内容。

AutoResponder 和 Composer 就不多做讲解了，主要因为 AutoResponder 作用更偏向于 Web 开发调试，在爬虫开发中实用性不强；Composer 虽然能够实现对服务器的请求，但在 Fiddler 编写代码就显得本末倒置，还不如直接使用 Python 实现。

4.7 Quickexec 命令行

Quickexec 命令行通过特定的条件快速找到符合条件的请求信息，如图 4-10 所示。

图 4-10 Quickexec 快速查找

从图 4-10 中看到，当输入“?gif”时，Web Session 列表会将符合条件的请求信息以高亮显示，在下方状态栏可以看到符合条件的请求有 44 条。

除了使用 Quickexec 实现快速查找之外，使用者还可以使用“Ctrl+F”查找功能，如图 4-11 所示。

通过输入关键字，然后单击查找按钮，就能找到相对应的 Session 信息，而且还可以自行设定目标高亮颜色。除此之外，使用者还能根据特定的要求设置不同的查找条件。

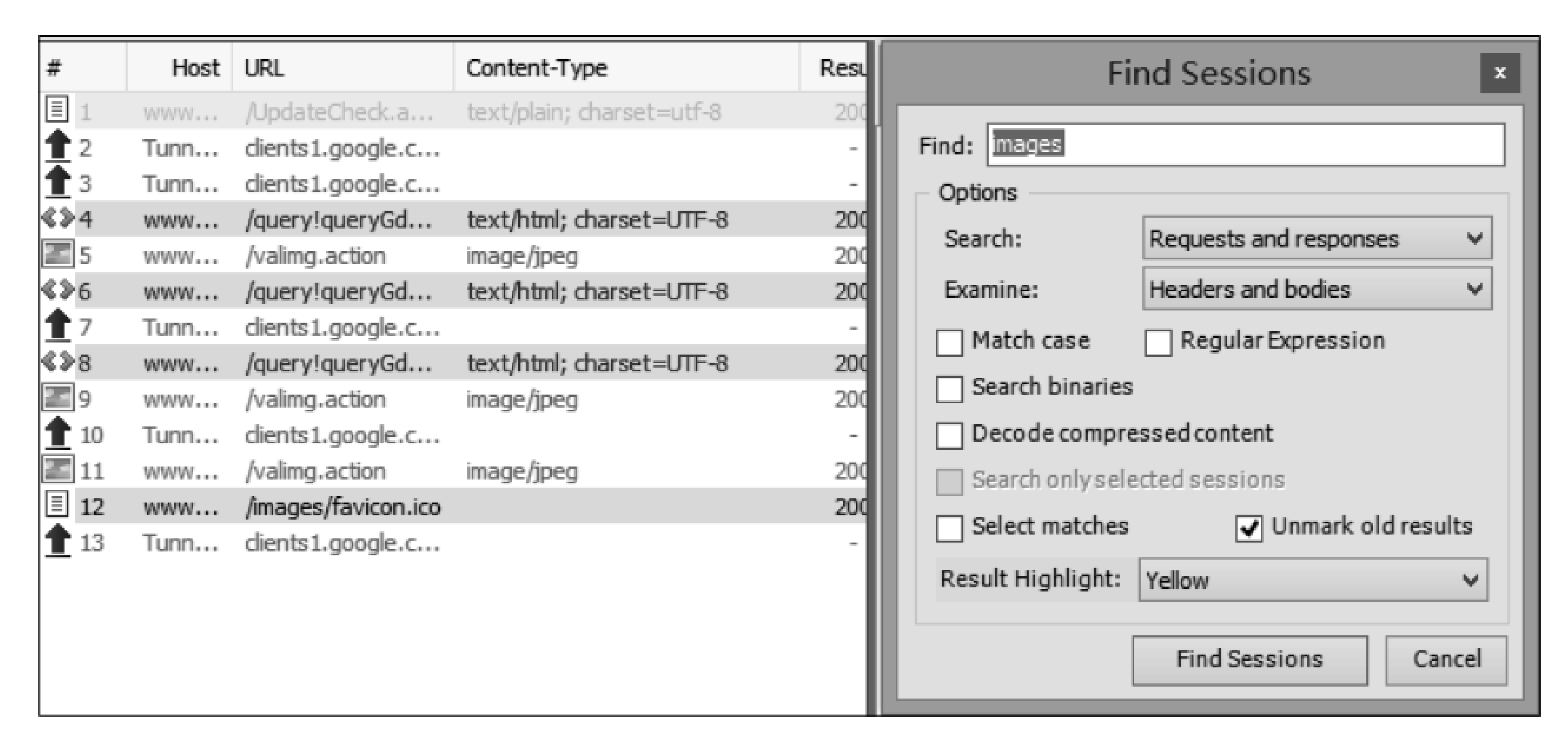


图 4-11 Ctrl+F 查找功能

4.8 本章小结

Fiddler 是一款非常流行并且实用的 HTTP 抓包工具，它的原理是在电脑中开启一个 HTTP 代理服务器，然后转发所有的 HTTP 请求和响应。因此，比一般的浏览器自带的抓包工具（开发者工具）要好用得多。不仅如此，Fiddler 还可以支持请求重放等一些高级功能，也可以支持对手机应用进行 HTTP 抓包。

Fiddler 提供了 Windows 环境下的 .exe 安装包，使其安装极其简单方便。安装完成后，需配置 HTTPS 抓取功能和手机抓包功能，完成配置便可对 HTTPS 网站和手机进行抓包。

除了功能强大之外，在使用上也较为简单，使用者只要打开 Fiddler，然后在浏览器（手机）中进行操作，Fiddler 就会自动抓取请求信息。就 Fiddler 本身的功能而言，使用者只需熟知每个功能按钮的作用便知道如何使用。

对于爬虫开发人员来说，需要掌握 Web Session 列表和 View 常用选项视图的基本功能，能够分析并得知每个请求的类型、状态码、请求方式、请求头、请求链接、请求参数以及响应内容等基本信息。

第 5 章

Urllib 数据抓取

5.1 Urllib 简介

Urllib 是 Python 自带的标准库，无须安装，直接引用即可。Urllib 通常用于爬虫开发、API（应用程序编程接口）数据获取和测试。在 Python 2 和 Python 3 中，Urllib 在不同版本中的语法有明显的改变。

Python 2 分为 Urllib 和 Urllib2，Urllib2 可以接收一个 Request 对象，并以此来设置一个 URL 的 Headers，但是 Urllib 只接收一个 URL，意味着不能伪装用户代理字符串等。Urllib 模块可以提供进行 Urlencode 的方法，该方法用于 GET 查询字符串的生成，Urllib2 不具有这样的功能。这也是 Urllib 与 Urllib2 经常在一起使用的原因。

在 Python 3 中，Urllib 模块是一堆可以处理 URL 的组件集合，就是将 Urllib 和 Urllib2 合并在一起使用，并且命名为 Urllib。

由于 Urllib 在不同的 Python 版本上有明显的区别，在实际开发中也遇到一些尴尬的情况，其中最为主要的是版本之间的互不兼容所带来的问题。

在 Python 3 中，Urllib 是一个收集几个模块来使用 URL 的软件包，大致具备以下功能。

- urllib.request：用于打开和读取 URL。
- urllib.error：包含提出的例外 urllib.request。
- urllib.parse：用于解析 URL。
- urllib.robotparser：用于解析 robots.txt 文件。

5.2 发送请求

urllib.request.urlopen 的语法如下：

```
urllib.request.urlopen(url, data=None, [timeout, ]*, cafile=None, capath=None, cadefault=False, context=None)
```

功能说明：Urllib 是用于访问 URL（请求链接）的唯一方法。

【参数解释】

- url：需要访问的网站的 URL 地址。url 格式必须完整，如 https://movie.douban.com/ 为完整的 url，若 url 为 movie.douban.com/，则程序运行时会提示无法识别 url 的错误。
- data：默认值为 None，Urllib 判断参数 data 是否为 None 从而区分请求方式。若参数 data 为 None，则代表请求方式为 GET；反之请求方式为 POST，发送 POST 请求。参数 data 以字典形式存储数据，并将参数 data 由字典类型转换成字节类型才能完成 POST 请求。
- timeout：超时设置，指定阻塞操作（请求时间）的超时（如果未指定，就使用全局默认超时设置）。
- cafile、capath 和 cadefault：使用参数指定一组 HTTPS 请求的可信 CA 证书。cafile 应指向包含一组 CA 证书的单个文件；capath 应指向证书文件的目录；cadefault 通常使用默认值即可。
- context：描述各种 SSL 选项的实例。

在实际使用中，常用的参数有 url、data 和 timeout。若在爬虫中遇到证书验证，则可将证书验证直接关闭，也可以设置参数指向证书的信息和位置。相比而言，设置证书比较耗时，而且通用性不强。

当对网站发送请求时，网站会返回相应的响应内容。urlopen 对象提供获取网站响应内容的方法函数，分别介绍如下。

- read()、readline()、readlines()、fileno() 和 close()：对 HTTPResponse 类型数据操作。
- info()：返回 HTTPMessage 对象，表示远程服务器返回的头信息。
- getcode()：返回 HTTP 状态码。
- geturl()：返回请求的 URL。

下面的例子用于实现 Urllib 模块对网站发送请求并将响应内容写入文本文档，代码如下：

```
# 导入 urllib
import urllib.request
# 打开 URL
response = urllib.request.urlopen('https://movie.douban.com/',
None, 2)
# 读取返回的内容
html = response.read().decode('utf8')
# 写入 txt
f = open('html.txt', 'w', encoding='utf8')
f.write(html)
f.close()
```

首先导入 urllib.request 模块，然后通过 urlopen 访问一个 URL，请求方式是 GET，所以参数 data 设置为 None；最后的参数用于设置超时时间，设置为 2 秒，如果超过 2 秒，网站还没返回响应数据，就会提示请求失败的错误信息。

当得到服务器的响应后，通过 response.read() 获取其响应内容。read() 方法返回的是一个 bytes 类型的数据，需要通过 decode() 来转换成 str 类型。最后将数据写入文本文档中，encoding 用于设置文本文档的编码格式，数据编码必须与文本文档编码一致，否则会出现乱码。运行结果如图 5-1 所示。

html.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
<!DOCTYPE html>
<html lang="zh-cmn-Hans" class="">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <meta name="renderer" content="webkit">
    <meta name="referrer" content="always">
    <title>
        豆瓣电影
</title>

    <meta name="baidu-site-verification" content="cZdR4xxR7RxmM4zE" />
    <meta http-equiv="Pragma" content="no-cache">
    <meta http-equiv="Expires" content="Sun, 6 Mar 2005 01:00:00 GMT">
```

图 5-1 获取豆瓣页面内容

5.3 复杂的请求

urllib.request.Request 的语法如下：

```
urllib.request.Request(url, data=None, headers={}, method=None)
```

功能说明：声明一个 request 对象，该对象可自定义 header（请求头）等请求信息。

【参数解释】

- url：完整的 url 格式，与 urllib.request.urlopen 的参数 url 一致。
- data：请求参数，与 urllib.request.urlopen 的参数 data 一致。
- headers：设置 request 请求头信息。
- method：设定请求方式，主要是 POST 和 GET 方式。

一个完整的 HTTP 请求必须要有请求头信息。而 urllib.request.Request 的作用是设置 HTTP 的请求头信息。使用 urllib.request.Request 为 5.2 节的例子设置请求头，代码如下：

```
# 导入urllib
import urllib.request
```

```
url = 'https://movie.douban.com/'
# 自定义请求头
headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko)'
    'Chrome/45.0.2454.85 Safari/537.36 115Browser/6.0.3',
    'Referer': 'https://movie.douban.com/',
    'Connection': 'keep-alive'}
# 设置 request 的请求头
req = urllib.request.Request(url, headers=headers)
# 使用 urlopen 打开 req
html = urllib.request.urlopen(req).read().decode('utf-8')
# 写入文件
f = open('html.txt', 'w', encoding='utf8')
f.write(html)
f.close()
```

5.4 代理 IP

代理 IP 的原理：以本机先访问代理 IP，再通过代理 IP 地址访问互联网，这样网站（服务器）接收到的访问 IP 就是代理 IP 地址。

Urllib 提供了 urllib.request.ProxyHandler() 方法可动态设置代理 IP 池，代理 IP 主要以字典格式写入方法。完成代理 IP 设置后，将设置好的代理 IP 写入 urllib.request.build_opener() 方法，生成对象 opener，然后通过 opener 的 open() 方法向网站（服务器）发送请求。

沿用前面章节的例子，将例子改为使用代理 IP 访问网站，代码如下：

```
import urllib.request
url = 'https://movie.douban.com/'
# 设置代理 IP
proxy_handler = urllib.request.ProxyHandler({
    'http': '218.56.132.157:8080',
    'https': '183.30.197.29:9797'})
# 必须使用 build_opener() 函数来创建带有代理 IP 功能的 opener 对象
```

```
opener = urllib.request.build_opener(proxy_handler)
response = opener.open(url)
html = response.read().decode('utf-8')
f = open('html.txt', 'w', encoding='utf8')
f.write(html)
f.close()
```

注意，由于使用代理 IP，因此连接 IP 的时候有可能出现超时而导致报错，遇到这种情况只要更换其他代理 IP 地址或者再次访问即可。以下是常见的报错信息。

- ConnectionResetError：[WinError 10054] 远程主机强迫关闭了一个现有的连接。
- urllib.error.URLError：urlopen error Remote end closed connection without response（结束没有响应的远程连接）。
- urllib.error.URLError: urlopen error [WinError 10054] 远程主机强迫关闭了一个现有的连接。
- TimeoutError: [WinError 10060] 由于连接方在一段时间后没有正确答复或连接的主机没有反应，因此连接尝试失败。
- urllib.error.URLError：urlopen error [WinError 10061] 由于目标计算机拒绝访问，因此无法连接。

5.5 使用 Cookies

Cookies 主要用于获取用户登录信息，比如，通过提交数据实现用户登录之后，会生成带有登录状态的 Cookies，这时可以将 Cookies 保存在本地文件中，下次程序运行的时候，可以直接读取 Cookies 文件来实现用户登录。特别对于一些复杂的登录，如验证码、手机短信验证登录这类网站，使用 Cookies 能简单解决重复登录的问题。

Urllib 提供 HTTPCookieProcessor() 对 Cookies 操作。但 Cookies 的读写是由 MozillaCookieJar() 完成的。下面的例子实现 Cookies 写入文件，代码如下：

```
import urllib.request
from http import cookiejar
filename = 'cookie.txt'
# MozillaCookieJar 保存 cookie
```

```
cookie = cookiejar.MozillaCookieJar(filename)
# HTTPCookieProcessor 创建 cookie 处理器
handler = urllib.request.HTTPCookieProcessor(cookie)
# 创建自定义 opener
opener = urllib.request.build_opener(handler)
# open 方法打开网页
response = opener.open('https://movie.douban.com/')
# 保存 cookie 文件
cookie.save()
```

代码中的 cookiejar 是自动处理 HTTP Cookie 的类，MozillaCookieJar() 用于将 Cookies 内容写入文件。程序运行时先创建 MozillaCookieJar() 对象，然后将对象直接传入函数 HTTPCookieProcessor()，生成 opener 对象；最后使用 opener 对象访问 URL，访问过程所生成的 Cookies 就直接写入已创建的文本文档中。

接着再看如何读取 Cookies，代码如下：

```
import urllib.request
from http import cookiejar
filename = 'cookie.txt'
# 创建 MozillaCookieJar 对象
cookie = cookiejar.MozillaCookieJar()
# 读取 cookie 内容到变量
cookie.load(filename)
# HTTPCookieProcessor 创建 cookie 处理器
handler = urllib.request.HTTPCookieProcessor(cookie)
# 创建 opener
opener = urllib.request.build_opener(handler)
# opener 打开网页
response = opener.open('https://movie.douban.com/')
# 输出结果
print(cookie)
```

读取和写入的方法很相似，主要区别在于：两者对 MozillaCookieJar() 对象的操作不同，导致实现功能也不同。运行结果如图 5-2 所示。

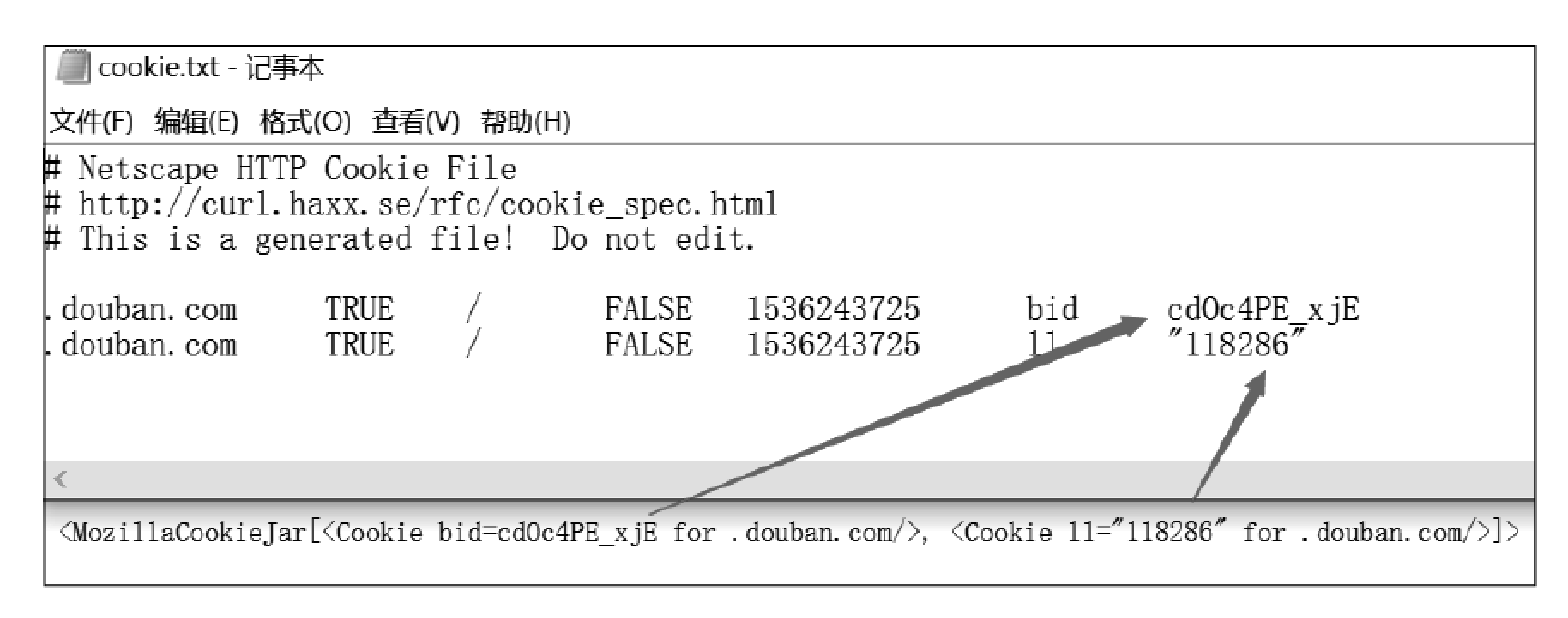


图 5-2 验证 Cookies

注意，为了方便测试，上述代码中使用的 cookie.save() 和 cookie.load(filename) 将 Cookies 内容显示在文本文档中。在实际开发中，为了提高安全性，可以在保存和读取 Cookies 时设置参数，使 Cookies 信息隐藏在文件中。方法如下：

```
cookie.save(ignore_discard=True,ignore_expires=True)
cookie.load(filename, ignore_discard=True, ignore_expires=True)
```

5.6 证书验证

当遇到一些特殊的网站时，在浏览器上会显示连接不是私密连接而导致无法浏览该网页。若在没有安装 12306 根证书的情况下访问 12306 网站，则页面如图 5-3 所示。



图 5-3 查询 12306 的车票

这里补充一个知识点，CA 证书也叫 SSL 证书，是数字证书的一种，类似于驾驶证、护照和营业执照的电子副本。因为配置在服务器上，也称为 SSL 服务器证书。

SSL 证书就是遵守 SSL 协议，由受信任的数字证书机构颁发 CA，在验证服务器身份后颁发，具有服务器身份验证和数据传输加密功能。

SSL 证书在客户端浏览器和 Web 服务器之间建立一条 SSL 安全通道（Secure Socket Layer，SSL），安全协议是由 Netscape Communication 公司设计开发的。该安全协议主要用来提供对用户和服务器的认证，对传送的数据进行加密和隐藏，确保数据在传送中不被改变，即数据的完整性，现已成为该领域中全球化的标准。

一些特殊的网站会使用自己的证书，如 12306 首页提示下载安装根证书，这是为了确保网站的数据在传输过程中的安全性。在讲述 urllib.request.urlopen 的时候，urlopen 带有 cafile、capath 和 cadefault 参数，可以用于设置用户的 CA 证书。

遇到这类验证证书的网站，最简单而暴力的方法是直接关闭证书验证，可以在代码中引入 SSL 模块，设置关闭证书验证即可。代码如下：

```
import urllib.request
import ssl
# 关闭证书验证
ssl._create_default_https_context = ssl._create_unverified_context
url = 'https://kyfw.12306.cn/otn/leftTicket/init'
response = urllib.request.urlopen(url)
# 输出状态码
print(response.getcode())
```

5.7 数据处理

我们知道 urllib.request.urlopen() 方法是不区分请求方式的，识别请求方式主要通过参数 data 是否为 None。如果向服务器发送 POST 请求，那么参数 data 需要使用 urllib.parse 对参数内容进行处理。

Urllib 在请求访问服务器的时候，如果发生数据传递，就需要对内容进行编码处理，将包含 str 或 bytes 对象的两个元素元组序列转换为百分比编码的 ASCII 文本字符串。如果字符串要用作 POST，那么它应该被编码为字节，否则会导致 TypeError 错误。

Urllib 发送 POST 请求的方法如下：

```
import urllib.request
import urllib.parse
url = 'https://movie.douban.com/'
data = {
    'value': 'true',
}
# 数据处理
data = urllib.parse.urlencode(data).encode('utf-8')
req = urllib.request.urlopen(url, data=data)
```

代码中 urllib.parse.urlencode(data) 将数据转换成字节的数据类型，而 encode('utf-8') 设置字节的编码格式。这里需要注意的是，编码格式主要根据网站的编码格式来决定。urlencode() 的作用只是对请求参数做数据格式转换处理。

除此之外，Urllib 还提供 quote() 和 unquote() 对 URL 编码处理，使用方法如下：

```
import urllib.parse
url = '%2523%25E7%25BC%2596%25E7%25A8%258B%2523'
# 第一次解码
first = urllib.parse.unquote(url)
print(first)
# 输出：'%23%E7%BC%96%E7%A8%8B%23'
# 第二次解码
second = urllib.parse.unquote(first)
print(second)
# 输出：'#编程#'
```

上述例子将已编码处理的 URL 进行解码还原，同样的方法，可使用 quote() 对数据进行编码处理。quote() 和 unquote() 的作用是解决请求参数中含有中文内容的问题。

5.8 本章小结

本章主要讲解了 Python 自带模块 Urllib 的功能和使用。Urllib 通常用于爬虫开发和 API（应用程序编程接口）数据获取和测试。在 Python 2 和 Python 3 中，Urllib 的

语法有明显的改变。其常用的语法有以下几种。

- urllib.request.urlopen：urllib 最基本的使用功能，用于访问 URL（请求链接）的唯一方法。
- urllib.request.Request：声明 request 对象，该对象可自定义请求头（header）、请求方式等信息。
- urllib.request.ProxyHandler：动态设置代理 IP 池，可加载请求对象。
- urllib.request.HTTPCookieProcessor：设置 Cookies 对象，可加载请求对象。
- urllib.request.build_opener()：创建请求对象，用于代理 IP 和 Cookies 对象加载。
- urllib.parse.urlencode(data).encode('utf-8')：请求数据格式转换。
- urllib.parse.quote(url)：URL 编码处理，主要对 URL 上的中文等特殊符号编码处理。
- urllib.parse.unquote(url)：URL 解码处理，将 URL 上的特殊符号还原。

除了 Urllib 之外，一些特殊请求需要结合其他模块配合使用，如 Cookies 读写由 HTTP 模块完成，关闭证书验证需要 SSL 模块设置，等等。

第 6 章

Requests 数据抓取

6.1 Requests 简介及安装

Requests 是 Python 的一个很实用的 HTTP 客户端库，完全满足如今网络爬虫的需求。与 Urllib 对比，Requests 是在 Urllib 的基础上进一步封装的，具备 Urllib 的全部功能；在开发使用上，语法简单易懂，完全符合 Python 优雅、简洁的特性；在兼容性上，完全兼容 Python 2 和 Python 3，具有较强的适用性。

Requests 可通过 pip 安装，具体如下。

- Windows 系统：pip install requests。
- Linux 系统：sudo pip install requests。

除了使用 pip 安装之外，还可以下载 whl 文件安装，方法如下：

（1）访问 www.lfd.uci.edu/~gohlke/pythonlibs，按 Ctrl+F 组合键搜索关键字“requests”，如图 6-1 所示。

requests-2.18.4-py2.py3-none-any.whl
requests_file-1.4.2-py2.py3-none-any.whl
requests_ftp-0.3.1-py2.py3-none-any.whl
requests_oauthlib-0.8.0-py2.py3-none-any.whl
requests_toolbelt-0.8.0-py2.py3-none-any.whl

图 6-1　安装 requests

（2）单击下载 requests-2.18.4-py2.py3-none-any.whl，把下载文件直接解压，将解压出来的文件直接放入 Python 的安装目录 Lib\site-packages 中即可。

（3）除了解压 whl，还可以使用 pip 安装 whl 文件。例如把下载的文件保存在 E 盘，打开 CMD（终端），将路径切换到 E 盘，输入安装命令：

```
E:\>pip install requests-2.18.4-py2.py3-none-any.whl
```

完成 Requests 安装后，在终端（CMD）下运行 Python，查看 Requests 版本信息，检测是否安装成功。方法如下：

```
E:\>python
>>> import requests
>>> requests.__version__
'2.18.4'
```

6.2 请求方式

HTTP 的常用请求是 GET 和 POST，Requests 对此区分两种不同的请求方式。GET 请求有两种形式，分别是不带参数和带参数，以百度为例：

```
# 不带参数
https://www.baidu.com/
# 带参数 wd
https://www.baidu.com/s?wd=python
```

判断 URL 是否带有参数，可以对符号“?”判断。一般网址末端（域名）带有“?”，就说明该 URL 是带有请求参数的，反之则不带有参数。GET 参数说明如下：

（1）wd 是参数名，参数名由网站（服务器）规定。

（2）python 是参数值，可由用户自行设置。

（3）如果一个 URL 有多个参数，参数之间用“&”连接。

Requests 实现 GET 请求，对于带参数的 URL 有两种请求方式：

```
import requests
# 第一种方式
r = requests.get('https://www.baidu.com/s?wd=python')
# 第二种方式
url = 'https://www.baidu.com/s'
params = {'wd': 'python'}
# 左边 params 在 GET 请求中表示设置参数
r = requests.get(url, params=params)
# 输出生成的 URL
print(r.url)
```

两种方式都是请求同一个 URL，在实际开发中建议使用第一种方式，因为代码简洁，如果参数是动态变化的，那么可使用字符串格式化对 URL 动态设置，例如 'https://www.baidu.com/s?wd=%s' %('python')。

POST 请求是我们常说的提交表单，表单的数据内容就是 POST 的请求参数。Requests 实现 POST 请求需设置请求参数 data，数据格式可以为字典、元组、列表和 JSON 格式，不同的数据格式有不同的优势。代码如下：

```
# 字典类型
data = {'key1': 'value1', 'key2': 'value2'}
# 元组或列表
(('key1', 'value1'), ('key1', 'value2'))
# JSON
import json
data = {'key1': 'value1', 'key2': 'value2'}
# 将字典转换 JSON
data=json.dumps(data)
```

```
# 发送 POST 请求
r = requests.post("https://www.baidu.com/", data=data)
print(r.text)
```

可以看出，左边的 data 是 POST 方法的参数，右边的 data 是发送请求到网站（服务器）的数据。值得注意的是，Requests 的 GET 和 POST 方法的请求参数分别是 params 和 data，别混淆两者的使用要求。

当向网站（服务器）发送请求时，网站会返回相应的响应（response）对象，包含服务器响应的信息。Requests 提供以下方法获取响应内容。

- r.status_code：响应状态码。
- r.raw：原始响应体，使用 r.raw.read() 读取。
- r.content：字节方式的响应体，需要进行解码。
- r.text：字符串方式的响应体，会自动根据响应头部的字符编码进行解码。
- r.headers：以字典对象存储服务器响应头，但是这个字典比较特殊，字典键不区分大小写，若键不存在，则返回 None。
- r.json()：Requests 中内置的 JSON 解码器。
- r.raise_for_status()：请求失败（非 200 响应），抛出异常。
- r.url：获取请求链接。
- r.cookies：获取请求后的 cookies。
- r.encoding：获取编码格式。

6.3 复杂的请求方式

从第 5 章得知，复杂的请求方式通常有请求头、代理 IP、证书验证和 Cookies 等功能。Requests 将这一系列复杂的请求做了简化，将这些功能在发送请求中以参数的形式传递并作用到请求中。

（1）添加请求头：请求头以字典的形式生成，然后发送请求中设置的 headers 参数，指向已定义的请求头，代码如下：

```
headers = {
    'content-type': 'application/json',
    'User-Agent': 'Mozilla/5.0 (Windows NT 6.3; WOW64;
                   rv:41.0) Gecko/20100101 Firefox/41.0'}
requests.get("https://www.baidu.com/", headers=headers)
```

（2）使用代理 IP：代理 IP 的使用方法与请求头的使用方法一致，设置 proxies 参数即可，代码如下：

```
import requests
proxies = {
  "http": "http://10.10.1.10:3128",
  "https": "http://10.10.1.10:1080",
}
requests.get("https://www.baidu.com/", proxies=proxies)
```

（3）证书验证：通常设置关闭验证即可。在请求设置参数 verify=False 时就能关闭证书的验证，默认情况下是 True。如果需要设置证书文件，那么可以设置参数 verify 值为证书路径。

```
import requests
url = 'https://kyfw.12306.cn/otn/leftTicket/init'
# 关闭证书验证
r = requests.get(url, verify=False)
print(r.status_code)
# 开启证书验证
# r = requests.get(url, verify=True)
# 设置证书所在路径
# r = requests.get(url, verify= '/path/to/certfile')
```

（4）超时设置：发送请求后，由于网络、服务器等因素，请求到获得相应会有一个时间差。如果不想程序等待时间过长或者延长等待时间，可以设定 timeout 的等待秒数，超过这个时间之后停止等待响应。如果服务器在 timeout 秒内没有应答，将会引发一个异常。使用代码如下：

```
requests.get("https://www.baidu.com/", timeout=0.001)
requests.post("https://www.baidu.com/", timeout=0.001)
```

（5）使用 Cookies：在请求过程中使用 Cookies 也只需设置参数 Cookies 即可。Cookies 的作用是标识用户身份，在 Requests 中以字典或 RequestsCookieJar 对象作为参数。获取方式主要是从浏览器读取和程序运行所产生。下面的例子进一步讲解如何使用 Cookies，代码如下：

```
import requests
temp_cookies='JSESSIONID_GDS=y4p7osFr_IYV5Udyd6c1drWE8MeTpQn0Y58T
g8cCONVP020y2N!450649273;name=value'
cookies_dict = {}
for i in temp_cookies.split(';'):
    value = i.split('=')
    cookies_dict [value[0]] = value[1]
r = requests.get(url, cookies=cookies)
print(r.text)
```

代码中变量 temp_cookies 是 Cookies 信息，可以在 Chrome 开发者工具→ Network →某请求的 Headers → Request Headers 中找到 Cookie 所对应的值。然后将字符串转换成字典格式，转换规则主要执行两次分割：第一次以“;”分割，得到列表 A，第二次是列表 A 的每一个元素以“=”分割，得到字典的键值对。

当程序发送请求时（不设参数 cookies），自动生成一个 RequestsCookieJar 对象，该对象用于存放 Cookies 信息。Requests 提供 RequestsCookieJar 对象和字典对象相互转换，代码如下：

```
import requests
url = 'https://movie.douban.com/'
r = requests.get(url)
# r.cookies 是 RequestsCookieJar 对象
print(r.cookies)
mycookies = r.cookies

# RequestsCookieJar 转换字典
cookies_dict = requests.utils.dict_from_cookiejar(mycookies)
print(cookies_dict)

# 字典转换 RequestsCookieJar
cookies_jar = requests.utils.cookiejar_from_dict(cookies_dict,
cookiejar=None, overwrite=True)
```

```
print(cookies_jar)

# 在 RequestsCookieJar 对象添加 Cookies 字典中
print(requests.utils.add_dict_to_cookiejar(mycookies, cookies_
dict))
```

如果要将 Cookies 写入文件，可使用 http 模块实现 Cookies 的读写。除此之外，还可以将 Cookies 以字典形式写入文件，此方法相比 http 模块读写 Cookies 更为简单，但安全性相对较低。使用方法如下：

```
import requests
url = 'https://movie.douban.com/'
r = requests.get(url)
# RequestsCookieJar 转换字典
cookies_dict = requests.utils.dict_from_cookiejar(mycookies)
# 写入文件
f = open('cookies.txt', 'w', encoding='utf-8')
f.write(str(cookies_dict))
f.close()
# 读取文件
f = open('cookies.txt', 'r')
dict_value = f.read()
f.close()
# eval(dict_value) 将字符串转换为字典
print(eval(dict_value))
r = requests.get(url, cookies=eval(dict_value))
print(r.status_code)
```

6.4 下载与上传

下载文件主要从服务器获取文件内容，然后将内容保存到本地。下载文件的方法如下：

```
import requests
url = 'http://cc.stream.qqmusic.qq.com/C100001Yyla31Dr60y.
m4a?fromtag=52'
```

```
r = requests.get(url)
f = open('mymusic.m4a', 'wb')
# r.content 获取响应内容（字节流）
f.write(r.content)
f.close()
```

代码变量 url 是一个音频文件 URL 地址，对文件所在 URL 地址发送请求（大多数是 GET 请求方式）；服务器将文件内容作为响应内容，然后将得到的内容以字节流（Bytes）格式写入自定义文件，这样就能实现文件下载。

除了文件下载外，还有更为复杂的文件上传，文件上传是将本地文件以字节流的方式上传到服务器，再由服务器接收上传内容，并做出相应的响应。文件上传存在一定的难度，其难点在于服务器接收规则不同，不同的网站，接收的数据格式和数据内容会不一致。下面以发送图片微博为例进行介绍。

（1）在浏览器中输入 https://weibo.cn/，在网页上单击“高级”按钮并使用 Fiddler 抓包工具（由于发送微博时，网页发生 302 跳转，因此使用 Chrome 会清空请求信息，导致抓取难度较大）。

（2）单击“选择文件”，选择图片文件并输入发布内容“Python 爬虫”，最后单击“发布”按钮发布微博。查看 Fiddler 抓取的请求信息，如图 6-2 所示。

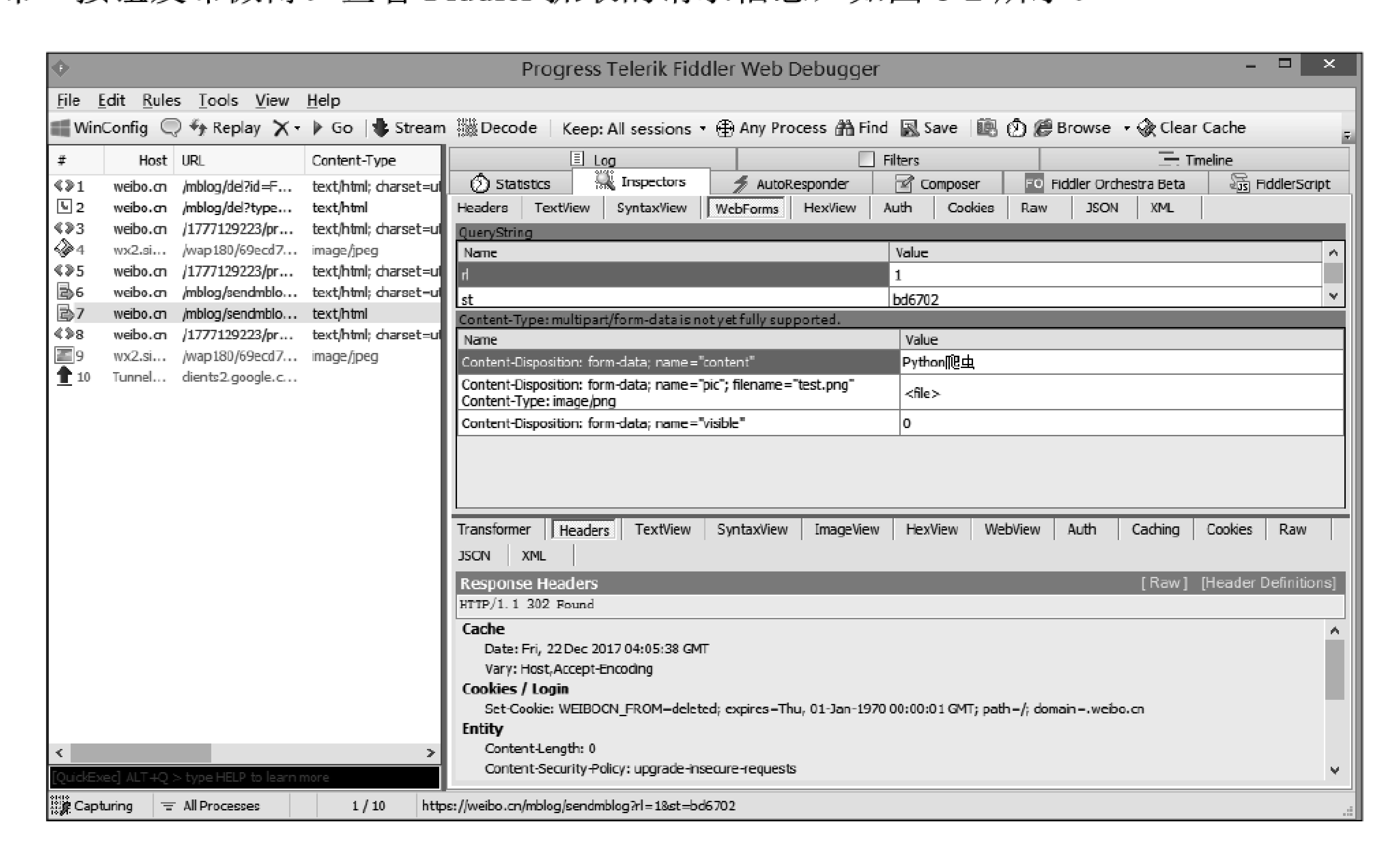


图 6-2 Fiddler 抓取的请求信息

从图 6-2 得知，该请求方式是 POST，QueryString 是 POST 的请求参数 data，Content-type 是上传文件，三个 Content-Disposition 分别对应发布内容、发布图片和设置分组可见。代码实现如下：

```
url = 'https://weibo.cn/mblog/sendmblog?rl=0&st=bd6702'
cookies = {'xxx': 'xxx'}
files = {'content': (None, 'Python 爬虫 '),
        'pic': ('pic', open('test.png', 'rb'),
        'image/png'),'visible': (None, '0')}
r = requests.post(url, files=files, cookies=cookies)
print(r.status_code)
```

POST 数据对象是以文件为主的，上传文件时使用 files 参数作为请求参数。Requests 对提交的数据和文件所使用的请求参数做了明确的规定。

参数 files 也是以字典形式传递的，每个 Content-Disposition 为字典的键值对，Content-Disposition 的 name 为字典的键，value 为字典的值。

此外，不同的网站设置对 files 参数的设置也是不一样的，下面列出较为常见的上传方法：

```
# 单独一个文件请求
{
  "field1" : open("filePath1", "rb").read())
}

# 同时选中多个文件
{
  "field1" : [
          ("filename1", open("filePath1", "rb")),
          ("filename2", open("filePath2", "rb"), "image/png"),
          open("filePath3", "rb"),
          open("filePath4", "rb").read()
          ]
}
```

6.5 本章小结

Requests 是 Python 的一个很实用的 HTTP 客户端库，可完全满足如今编写网络爬虫程序的需求，是爬虫开发人员首选的爬虫库。其具有语法简单易懂，完全符合 Python 优雅和简洁的特性，在兼容性上完全兼容 Python 任何版本，具有较强的适用性。

读者要掌握 Requests 实现 GET 和 POST 请求是分别使用了不同的方法，如下：

```
import requests
url = 'https://baidu.com/'
# GET 请求
r = requests.get(url, headers=headers, proxies=proxies,
verify=False, cookies=cookies)
# POST 请求
r = requests.post(url, data=data, files=files, headers=headers,
proxies=proxies,
                    verify=False, cookies=cookies)
```

Requests 的 GET 和 POST 将请求中所需要使用的功能都以参数的形式直接作用到请求中。一个发送请求的语句就已包含了请求头、代理 IP、Cookies、证书验证、文件上传等功能。

另外，Requests 还提供了 r.status_code、r.raw、r.content、r.text、r.headers、r.json()、r.raise_for_status()、r.url、r.cookies、r.encoding 十种方法获取响应内容。

第 7 章 验证码识别

7.1 验证码类型

在开发爬虫时，经常会遇到验证码识别，在网站中加入验证码的目的是加强用户安全性和提高反爬虫机制，有效防止对某一个特定注册用户用特定程序暴力破解的方式不断地进行登录尝试。在此简单地为大家介绍一下验证码的种类。

- 字符验证码：在图片上随机产生数字、英文字母或汉字，一般有 4 位或者 6 位验证码字符。通过添加干扰线、添加噪点以及增加字符的粘连程度和旋转角度来增加机器识别的难度。但是这种传统的验证码随着 OCR 技术的发展，能够轻易地被破解。

- 图片验证码：图片验证码也只是换汤不换药，应用了字符验证码的技术，只是不是随机的字符，而是让人识别图片，比如 12306 的验证码。同时还包括一些将广告嵌入图片上面的验证码，都应该归属到这一类。
- GIF 动画验证码：主流验证码提供静态的图片，比较容易被 OCR 软件识别，有的网站提供 GIF 动态的验证码图片，使得识别器不容易辨识哪一个图层是真正的验证码图片，在提供清晰图片的同时，可以更有效地防止识别器的识别。据统计，动画 GIF 验证码的防垃圾注入可以达到 100%，是一个非常有效的验证码创新模式。同时，GIF 动画效果多达百种，也可以增加网站页面的美观效果。
- 极验验证码：是极验验证于 2012 年推出的新型验证码，基于行为式验证技术，通过拖动滑块完成拼图的形式实现验证，是目前看到的比较有创意的验证码，安全性具有新的突破。
- 手机验证码：通过短信的形式发送到用户手机上面的验证码，一般为 6 位的数字。
- 语音验证码：也属于手机端验证的一种方式。
- 视频验证码：视频验证码是验证码中的新秀，在视频验证码中，将随机数字、字母和中文组合而成的验证码动态嵌入 MP4、FLV 等格式的视频中，增大破解难度。验证码视频动态变换、随机响应，可以有效防范字典攻击、穷举攻击等攻击行为。视频中的验证码字母、数字组合，字体的形状、大小，速度的快慢，显示效果和轨迹的动态变换，增加了恶意抓屏破解的难度。其安全度远高于普通的验证码，而且这种验证码形式使用户不会感到枯燥，由于其提高了机器识别的难度，因此可以降低用户识别的难度，使得用户更容易辨认。

现在大多数网站还使用字符验证码，主要用于用户登录。有些网站数据需要用户登录才有访问权限，爬取这样的数据时，首先要完成用户登录，获取访问权限才能继续下一步的数据爬取。

对于用户登录设置验证码识别的网站有三种解决方案：

（1）人工识别验证码。将验证码图片下载到本地，然后靠使用者自行识别并将识别内容输入，程序获取输入内容，完成用户登录。其特点是开发简单，适合初学者，但过分依赖人为控制，难以实现批量爬取。

（2）通过 Python 调用 OCR 引擎识别验证码。这是最理想的解决方案，但正常情况下，OCR 准确率较低，需要机器学习不断提高 OCR 准确率，开发成本相对较高。

（3）调用 API 使用第三方平台识别验证码。开发成本较低，有完善的 API 接口，直接调用即可，识别准确率高，但每次识别需收取小额费用。

上述方案是目前解决验证码最有效的手段，本章主要介绍如何使用 OCR 技术和第三方平台识别验证码。

7.2 OCR 技术

OCR（Optical Character Recognition，光学字符识别）是指电子设备（例如扫描仪或数码相机）检查纸上打印的字符，通过检测暗、亮的模式确定其形状，然后用字符识别方法将形状翻译成计算机文字的过程，即针对印刷体字符，采用光学的方式将纸质文档中的文字转换成为黑白点阵的图像文件，并通过识别软件将图像中的文字转换成文本格式，供文字处理软件进一步编辑加工的技术。

在 Python 中，支持 ORC 的模块有 pytesser3 和 pyocr，其原理主要是通过模块功能调用 OCR 引擎识别图片，OCR 引擎再将识别的结果返回到程序中。本节主要以 pyocr 为例进行介绍。在 Windows 中安装 pyocr 可以在 CMD 下使用 pip 安装：

```
pip install pyocr
```

安装 pyocr 模块之后，还需要安装 PIL 模块，这是专门用于处理图片的模块，pyocr 依赖该模块才能完成识别，pip 安装如下：

```
pip install Pillow
```

完成 pyocr 和 PIL 模块的安装后，最后是 OCR 引擎的安装，图像识别主要由 OCR 引擎完成，pyocr 只起到一个调用引擎的作用。

Tesseract-OCR 是一个免费、开源的 OCR 引擎，读者可从网上自行搜索下载安装。在 Windows 系统中，OCR 引擎（Tesseract-OCR）可通过 .exe 安装包安装。值得注意的是，在安装过程中有附加功能选项，如图 7-1 所示。

选项“Tesseract development files”是 OCR 开发文件，可以在这个引擎的基础上进行二次开发。若安装时勾选该选项，则安装过程中会访问谷歌服务器下载开发文件。

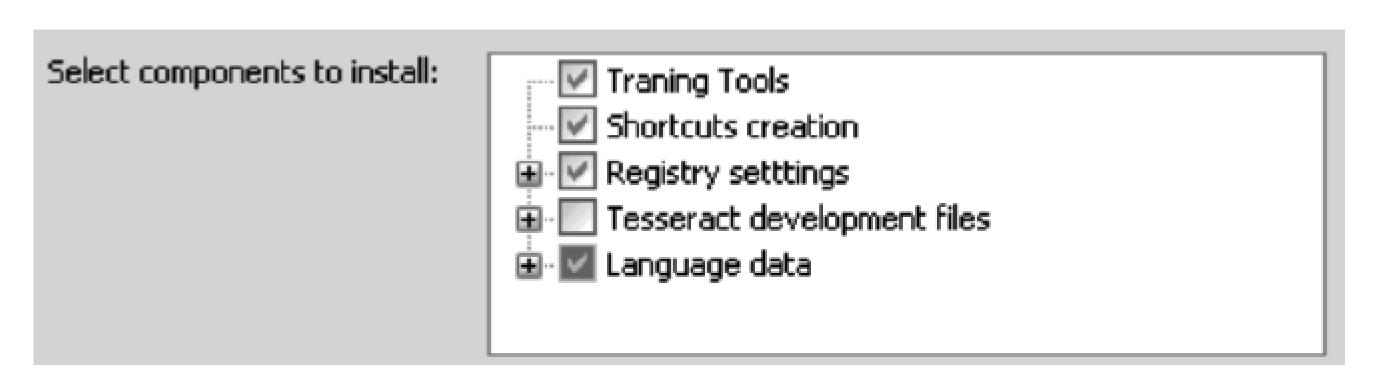


图 7-1　OCR 安装选项

选项“Language data”默认勾选英文，这是识别文字选项。如果要识别其他国家的语言，可自行勾选，但勾选其他语言，在下一步安装时需要访问谷歌下载文件。除此之外，可自行下载 chi_sim.traineddata 文件（中文简体语言包），然后放到 C:\Program Files (x86)\Tesseract-OCR\tessdata 文件夹下即可（上述路径是 OCR 引擎默认的安装路径）。

完成上述安装后，就能在 Python 中使用 pyocr 实现 OCR 识别了，方法如下：

（1）创建 OCR 文件夹，在该文件夹下创建 ocr.py 文件和图片 pic.png，如图 7-2 所示。

（2）打开 ocr.py 文件，输入代码：

```
from PIL import Image
from pyocr import tesseract
# 使用 PIL 打开图片
im = Image.open('pic.png')
# OCR 识别
code = tesseract.image_to_string(im)
print(code)
```

（3）运行 ocr.py，运行结果如图 7-3 所示。



图 7-2　OCR 使用

```
E:\Python\python.exe E:/OCR/orc.py
2334234

Process finished with exit code 0
```

图 7-3　验证码识别结果

在实际使用时，验证码图片不会是一张白底黑字的图片，往往会掺入很多干扰因

素，这样会导致识别出来的结果与实际相差甚大。为了提高准确率，可以使用 PIL 模块对图片进行简单的处理，如图 7-4 所示。

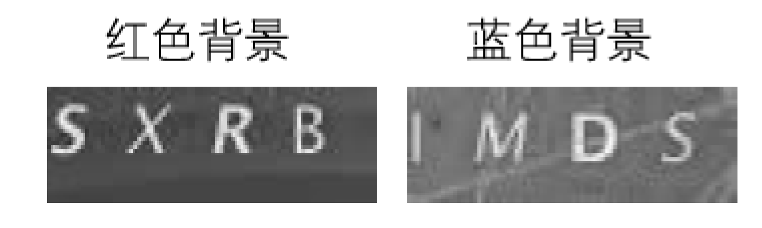


图 7-4 验证码图片

图 7-4 分别是带红色和蓝色背景的验证码，而且背景颜色带有其他杂色，如果使用上述代码对图片进行识别，识别结果就与实际完全不相符。因此，我们可以对图片做简单的处理，去掉干扰因素，提高识别准确率。图片处理主要由 PIL 模块实现，图 7-4 中的验证码分别命名为 pic1.png 和 pic2.png 文件，实现代码如下：

```
from PIL import Image
from pyocr import tesseract

pic_list = ['pic1.png','pic2.png']
for i in pic_list:
    im = Image.open(i)
    im = im.convert('L')# 图片转换为灰色图像
    # 保存转换后的图片
    im.save("temp.png")
    code = tesseract.image_to_string(im)
    print(code)
```

PIL 模块打开图片并生成图片对象 im，然后转换图片颜色模式，将带有颜色的图片转换成灰度模式，形成黑→灰→白的过渡，如同黑白照片，最后交给 OCR 引擎识别并返回识别结果。

补充说明：颜色模式是将某种颜色表现为数字形式的模型，或者是一种记录图像颜色的方式，分为 RGB 模式、CMYK 模式、HSB 模式、Lab 颜色模式、位图模式、灰度模式、索引颜色模式、双色调模式和多通道模式。

程序运行结果如图 7-5 所示。

```
E:\Python\python.exe E:/OCR/orc.py
IMDS
SXRB

Process finished with exit code 0
```

图 7-5 验证码识别结果

本书只介绍简单的图片处理，不同的图片有不同的处理方法，其目的是提高 OCR 识别的准确率。除此之外，提高 OCR 准确率还可以对 OCR 引擎进行训练和学习。但两者已经属于人工智能的领域，其涉及较多的知识点，所以就不一一讲解了。

7.3 第三方平台

除了使用 OCR 识别验证码之外，还可以利用第三方平台实现验证码的识别。到目前为止，这是解决验证码最快、最简单的途径，而且有完善的 API 接口，能帮助开发者完成快速开发的需求，但每次调取 API 接口需要收取少量费用。

验证码识别平台主要有：

- 打码平台：主要由在线人员识别验证码。开发者只需调用平台 API 接口，一般在 10 秒内返回结果。识别错误或者无法识别不收费。
- AI 开发者平台：主要由人工智能系统识别，准确率取决于系统的智能程度。调用 API 接口每天有免费使用次数，也可以付费使用。目前，主流平台有百度 AI 和腾讯 AI。

本书以打码平台为例，在浏览器上访问 http://www.yundama.com/，注册账号后充值就能调用 API 接口识别验证码，在平台上提供开发文档，代码如下：

```
import json
import time
import requests
class YDMHttp:
    apiurl = 'http://api.yundama.com/api.php'
    username = ''
    password = ''
    appid = '4055'
    appkey = 'c5e26d1a207df586d7aaec21522dd446'
    def __init__(self, name, passwd, app_id, app_key):
        self.username = name
        self.password = passwd
        self.appid = str(app_id)
        self.appkey = app_key
```

```
    def request(self, fields, files=[]):
        response = self.post_url(self.apiurl, fields, files)
        response = json.loads(response)
        return response

    def balance(self):
        data = {
            'method': 'balance',
            'username': self.username,
            'password': self.password,
            'appid': self.appid,
            'appkey': self.appkey
        }
        response = self.request(data)
        if response:
            if response['ret'] and response['ret'] < 0:
                return response['ret']
            else:
                return response['balance']
        else:
            return -9001

    def login(self):
        data = {'method': 'login', 'username': self.username,
               'password': self.password, 'appid': self.appid,
               'appkey': self.appkey}
        response = self.request(data)
        if response:
            if response['ret'] and response['ret'] < 0:
                return response['ret']
            else:
                return response['uid']
        else:
            return -9001

    def upload(self, filename, codetype, timeout):
```

```
        data = {'method': 'upload', 'username': self.username,
    'password': self.password, 'appid': self.appid,
                'appkey': self.appkey, 'codetype': str(codetype),
'timeout': str(timeout)}
        file = {'file': filename}
        response = self.request(data, file)
        if response:
            if response['ret'] and response['ret'] < 0:
                return response['ret']
            else:
                return response['cid']
        else:
            return -9001

    def result(self, cid):
        data = {'method': 'result', 'username': self.username,
    'password': self.password, 'appid': self.appid,
                'appkey': self.appkey, 'cid': str(cid)}
        response = self.request(data)
        return response and response['text'] or ''

    def decode(self, file_name, code_type, time_out):
        cid = self.upload(file_name, code_type, time_out)
        if cid > 0:
            for i in range(0, time_out):
                result = self.result(cid)
                if result != '':
                    return cid, result
                else:
                    time.sleep(1)
            return -3003, ''
        else:
            return cid, ''

    def post_url(self, url, fields, files=[]):
        for key in files:
            files[key] = open(files[key], 'rb')
```

```
        res = requests.post(url, files=files, data=fields)
        return res.text

def code_verificate(name, passwd, file_name, app_id=4055, code_
type=1005, time_out=60):
    # name: 云打码注册用户名; passwd: 用户密码; file_name: 需要识别的图片名
    app_key='c5e26d1a207df586d7aaec21522dd446'
    yundama_obj = YDMHttp(name, passwd, app_id, app_key)
    cur_uid = yundama_obj.login()
    print('uid: %s' % cur_uid)
    rest = yundama_obj.balance()
    print('balance: %s' % rest)
    # 开始识别图片路径、验证码类型 ID、超时时间（秒），并显示识别结果
    cid, result = yundama_obj.decode(file_name, code_type, time_
out)
    print('cid: %s, result: %s' % (cid, result))
    return result
if __name__ == '__main__':
    # 云打码注册的登录用户名（通过用户注册）
    username = 'xxx'
    # 登录密码
    password = 'xxx'
    rs = code_verificate(username, password, 'pincode.png')
```

使用方法：只需在爬虫代码中引用上述文档，然后调用 code_verificate() 方法函数，传入已注册的用户信息和需要识别的图片即可。

7.4 本章小结

本章中读者应重点掌握以下内容。

1. 验证码

验证码的作用是加强用户安全性和提高反爬虫机制，有效防止这种问题对某一个特定注册用户用特定程序暴力破解的方式不断地进行登录尝试。

读者要了解解决验证码的以下几种方案：

（1）人工识别验证码，将验证码图片下载到本地，然后靠使用者自行识别并输入识别内容，程序获取输入的内容后，用户完成登录。其特点是开发简单，适合初学者，但过分依赖人为控制，难以实现批量爬取。

（2）通过 Python 调用 OCR 引擎识别验证码。这是最理想的解决方案，但 OCR 准确率较低，需要机器学习不断提高 OCR 的准确率，开发成本相对较高。

（3）调用 API 使用第三方平台识别验证码。开发成本较低，有完善的 API 接口，直接调用即可，识别准确率高，但每次识别需收取小额费用。

2. OCR

OCR （Optical Character Recognition，光学字符识别）是指使用电子设备（例如扫描仪或数码相机）检查纸上打印的字符，通过检测暗、亮的模式确定其形状，然后用字符识别方法将形状翻译成计算机文字的过程；即针对印刷体字符，采用光学的方式将纸质文档中的文字转换成为黑白点阵的图像文件，并通过识别软件将图像中的文字转换成文本格式，供文字处理软件进一步编辑加工的技术。

Python 中支持的 ORC 模块有 pytesser3 和 pyocr，其原理主要是通过模块功能调用 OCR 引擎识别图片，OCR 引擎再将识别的结果返回到程序中。

3. 验证码识别平台

验证码识别平台主要有以下两种。

（1）打码平台：主要由在线人员识别验证码。开发者只需调用平台 API 接口，一般在 10 秒内返回结果。识别错误或者无法识别不收费。

（2）AI 开发者平台：主要由人工智能系统识别，准确率取决于系统的智能程度。调用 API 接口使用，每天有免费使用次数，也可付费使用，目前主流平台有百度 AI 和腾讯 AI。

第 8 章

数据清洗

8.1 字符串操作

从网页上采集数据后，数据大多数是杂乱无章的，这时需要对采集的数据加工清洗，去掉数据中的一些垃圾数据才能得到我们所需的数据。清洗数据有三种常用的方法：字符串操作、正则表达式和第三方模块库。三种方法在不同场景有不同优势，取长补短，应根据实际情况选择合理的清洗方法，三种方法同时出现在一个项目也是常见的事情。

用于清洗数据的字符串操作：截取、替换、查找和分割。

（1）截取：字符串 [开始位置 : 结束位置 : 间隔位置]。开始位置是 0，正数代表从左边位置开始，负数代表从右边位置开始，默认代表从 0 开始。结束位置是被截

取的字符串位置，空值默认取到字符串尾部。间隔位置默认为1，截取的内容不做处理；如果设置为2，就将截取的内容再隔一取数。例子如下：

```
# 字符串截取
str = 'ABCDEFG'
# 截取第一位到第三位的字符
print(' 截取第一位到第三位的字符 : ' + str[0:3:])
# 截取字符串的全部字符
print(' 截取字符串的全部字符 : ' + str[::])
# 截取第七个字符到结尾
print(' 截取第七个字符到结尾 : ' + str[6::])
# 截取从头开始到倒数第三个字符之前
print(' 截取从头开始到倒数第三个字符之前 : ' + str[:-3:])
# 截取第三个字符
print(' 截取第三个字符 : ' + str[2])
# 截取倒数第一个字符
print(' 截取倒数第一个字符 : ' + str[-1])
# 与原字符串顺序相反的字符串
print(' 与原字符串顺序相反的字符串 : ' + str[::-1])
# 截取倒数第三位与倒数第一位之前的字符
print(' 截取倒数第三位与倒数第一位之前的字符 : ' + str[-3:-1:])
# 截取倒数第三位到结尾
print(' 截取倒数第三位到结尾 : ' + str[-3::])
# 逆序截取
print(' 逆序截取 : ' + str[:-5:-3])
```

（2）替换：字符串.replace('被替换内容','替换后内容')。要注意的是，使用replace替换字符串后仅为临时变量，需重新赋值才能保存。例子如下：

```
str = 'ABCABCABC'
# 单个内容替换
print(str.replace('C', 'V'))
# 输出内容：ABVABVABV

# 字符串替换
print(str.replace('BC', 'WV'))
# 输出内容：AWVAWVAWV
# 替换成特殊符号（空格）
```

```
print(str.replace('BC', ' '))
# 输出内容：A A A
```

（3）查找：字符串 .find(' 要查找的内容 ' [, 开始位置 , 结束位置])，开始位置和结束位置表示要查找的范围，若为空值，则表示查找所有。找到目标后会返回目标第一位内容所在的位置，位置从 0 开始算，如果没找到，就返回 -1。例子如下：

```
str = 'ABCDABC'
# 查找全部
print(str.find('A'))
# 输出内容：0

# 从字符串第 4 个开始查找
print(str.find('A', 3))
# 输出内容：4

# 从字符串第 2 个到第 6 个开始查找，即从 'BCDAB' 中查找 'C'
print(str.find('C', 1, 5))
# 输出内容：2

# 查找不存在的内容
print(str.find('E'))
# 输出内容：-1
```

除了使用 find 函数查找字符串中某个内容之外，index 函数也能实现同样的功能。index 是在字符串里查找子串第一次出现的位置，类似于字符串的 find 方法，如果查找不到子串，就会抛出异常，而不是返回 -1。例子如下：

```
str = 'ABCDABC'
# 查找全部
print(str.index('A'))
# 输出内容：0

# 从字符串第 4 个开始查找
print(str.index('A', 3))
# 输出内容：4
```

```
# 从字符串第 2 个到第 6 个开始查找
print(str.index('C', 1, 5))
# 输出内容：2

# 查找不存在的内容
print(str.index('E'))
# 输出内容：ValueError: substring not found
```

（4）分割：字符串 .split('分割符', 分割次数)。如果存在分割次数，就仅分割成“分割次数+1”个子字符串；如果为空，就默认全部分割。分割后，返回一个列表类型数据。例子如下：

```
str = 'ABCDABC'
# 分割全部
print(str.split('B'))
# 输出内容：['A', 'CDA', 'C']
# 分割一次
print(str.split('B', 1))
# 输出内容：['A', 'CDABC']
```

字符串操作是数据清洗的基本，可以解析 HTML，但纯字符串解析 HTML 会导致代码冗余，不便维护，一般不建议这样操作。字符串操作主要用于个别数据清洗，且数据具有一定的特性。如图 8-1 所示是一个例子。

```
← → C  ▲ 不安全 | https://kyfw.12306.cn/otn/resources/js/framework/station_name.js?station_version=1.9031

var station_names ='@bjb|北京北|VAP|beijingbei|bjb|0@bjd|北京东|BOP|beijingdong|bjd|1@bji|北京|BJP|beijing|bj|2@bjn|北京南|V
南|IZQ|guangzhounan|gzn|5@cqb|重庆北|CUW|chongqingbei|cqb|6@cqi|重庆|CQW|chongqing|cq|7@cqn|重庆南|CRW|chongqingnan|cqn|8@gz
海南|SNH|shanghainan|shn|11@shq|上海虹桥|AOH|shanghaihongqiao|shhq|12@shx|上海西|SXH|shanghaixi|shx|13@tjb|天津北|TBP|tianjin
南|TIP|tianjinnan|tjn|16@tjx|天津西|TXP|tianjinxi|tjx|17@cch|长春|CCT|changchun|cc|18@ccn|长春南|CET|changchunnan|ccn|19@ccx
东|ICW|chengdudong|cdd|21@cdn|成都南|CNW|chengdunan|cdn|22@cdu|成都|CDW|chengdu|cd|23@csh|长沙|CSQ|changsha|cs|24@csn|长沙南
南|FYS|fuzhounan|fzn|27@gya|贵阳|GIW|guiyang|gy|28@gzh|广州|GZQ|guangzhou|gz|29@gzx|广州西|GXQ|guangzhouxi|gzx|30@heb|哈尔滨
西|VAB|haerbinxi|hebx|33@hfe|合肥|HFH|hefei|hf|34@hfx|合肥西|HTH|hefeixi|hfx|35@hhd|呼和浩特东|NDC|huhehaotedong|hhhtd|36@hh
东|KEQ|haikoudong|hkd|38@hkd|海口东|HMQ|haikoudong|hkd|39@hko|海口|VUQ|haikou|hk|40@hzd|杭州东|HGH|hangzhoudong|hzd|41@hzh|杭
南|JNK|jinan|jn|44@jnd|济南东|JAK|jinandong|jnd|45@jnx|济南西|JGK|jinanxi|jnx|46@kmi|昆明|KMM|kunming|km|47@kmx|昆明西|KXM|k
东|LVJ|lanzhoudong|lzd|50@lzh|兰州|LZJ|lanzhou|lz|51@lzx|兰州西|LAJ|lanzhouxi|lzx|52@nch|南昌|NCG|nanchang|nc|53@nji|南京|NJ
```

图 8-1 12306 各个城市的站点信息

在浏览器中输入“https://kyfw.12306.cn/otn/leftTicket/init”，在开发者工具→ Network → JS 标签中可找到图 8-1 中的站点信息。根据内容分析，每个城市有 5 个信息，从带有“@”的特殊字符开始，每个信息之间用“|”隔开。如果想要获取第二个和第三个信息，可以根据其特性以“|”进行字符串分割，代码如下：

```
import requests
def city_name():
    # 构建请求头
    headers = {'User-Agent':
                   'Mozilla/5.0 (Windows NT 10.0; WOW64)
AppleWebKit/537.36 '
                   '(KHTML, like Gecko) Chrome/63.0.3218.0
Safari/537.36',
                   'Referer':
                   'https://kyfw.12306.cn/otn/login/init'}
    url = 'https://kyfw.12306.cn/otn/resources/js/framework/
station_name.js?station_version=1.9031'
    city_code = requests.get(url, headers=headers, verify=False)
    # 数据使用字符串操作处理
    city_code_list = city_code.text.split("|")
    city_dict = {}
    for k, i in enumerate(city_code_list):
        if '@' in i:
# 城市名作为字典的键，城市编号作为字典的值
            city_dict[city_code_list[k + 1]] = city_code_list[k +
2].replace(' ', '')
    return (city_dict)
# 输出处理后的数据
print(city_name())
```

除了使用 split 对字符串进行分割之外，在数据赋值之前还要使用 replace() 对数据进行替换，主要清洗数据中含有空白的内容。在一些设计不规范的网站中，其 HTML 中的数据经常带有空白内容和一些特殊符号，可以使用 replace() 对这类数据进行清洗。

8.2 正则表达式

正则表达式是用于处理字符串的强大工具，拥有自己独特的语法以及一个独立的处理引擎，效率上可能不如字符串处理方法，但功能十分强大。得益于这一点，在提供了正则表达式的语言里，正则表达式的语法都是一样的，区别只在于不同的编程语言实现支持的语法数量不同，但不用担心，不被支持的语法通常是不常用的部分。

学习正则表达式要从两方面着手：正则语法和正则处理函数。

正则语法：也称元字符，符合正则规则，通常表示一些不寻常的匹配操作，或者通过重复、修改匹配意义来影响正则模式的其他部分。常用语法如表 8-1 所示。

表8-1　正则表达式元字符和语法

元字符	说　明	实　例
.	匹配任意字符（不包括换行符）	'abc'>>>'a.c'>>>结果为：'abc'
^	匹配开始位置，多行模式下匹配每一行的开始	'abc'>>>'^abc'>>>结果为：'abc'
$	匹配结束位置，多行模式下匹配每一行的结束	'abc'>>>'abc$'>>>结果为：'abc'
*，+，？	匹配前一个元字符0到多次	'abcccd'>>>'abc*'>>>结果为：'abccc'
+	匹配前一个元字符1到多次	'abcccd'>>>'abc+'>>>结果为：'abccc'
?	匹配前一个元字符0到1次	'abcccd'>>>'abc?'>>>结果为：'abc'
{m}	匹配前一个字符m次	'abcccd'>>>'abc{3}d'>>>结果为：'abcccd'
{m,n}	匹配前一个字符m到n次	'abcccd'>>>'abc{2,3}d'>>>结果为：'abcccd'
{m,n}?	匹配前一个字符 m 到 n 次，并且取尽可能少的情况	'abccc'>>>'abc{2,3}?'>>>结果为：'abcc'
\\	对特殊字符进行转义，或者指定特殊序列	'a.c'>>>'a\\.c'>>> 结果为：'a.c'
[]	字符集，一个字符的集合，可匹配其中任意一个字符	'abcd'>>>'a[bc]'>>>结果为：'ab'
|	逻辑表达式“或”，比如 a|b 代表可匹配 a 或者 b	'abcd'>>>'abc|acd'>>>结果为：'abc'
(…)	被括起来的表达式作为一个分组。findall 在有组的情况下只显示组的内容	'a123d'>>>'a(123)d'>>>结果为：'123'
(?#...)	添加注释，括号内为注释内容，特殊构建不作为分组	'abc123'>>>'abc(?#fasd)123'>>>结果为：'abc123'
(?= …)	顺序肯定环视，表示所在位置右侧能够匹配括号内正则	在字符串'pythonretest'中(?=test)会匹配'pythonre'
(?!...)	顺序否定环视，表示所在位置右侧不能匹配括号内正则	如果'pythonre'右侧不是字符串' test'，也就是说字符串为testpythonre，那么(?!test)会匹配' pythonre'
(?<= …)	逆序肯定环视，表示所在位置左侧能够匹配括号内正则	与(?!...)实例一致
(?<!...)	逆序否定环视，表示所在位置左侧不能匹配括号内正则	与(?= …)实例一致

正则表达式特殊序列说明如表 8-2 所示。

表8-2 正则表达式特殊序列

特殊表达式序列	说　明
\A	只在字符串开头进行匹配
\b	匹配位于开头或者结尾的空字符串
\B	匹配不位于开头或者结尾的空字符串
\d	匹配任意十进制数，相当于 [0-9]
\D	匹配任意非数字字符，相当于 [^0-9]
\s	匹配任意空白字符，相当于 [\t\n\r\f\v]
\S	匹配任意非空白字符，相当于 [^ \t\n\r\f\v]
\w	匹配任意数字和字母，相当于 [a-zA-Z0-9_]
\W	匹配任意非数字和字母的字符，相当于 [^a-zA-Z0-9_]
\Z	只在字符串结尾进行匹配

正则处理函数：Python 的正则模块是 re，该模块含有多种正则处理函数，常用的功能函数包括：match、search、findall 和 sub。

（1）re.match(pattern, string, flags=0)，re.match 函数尝试从字符串的开头开始匹配一个模式，如果匹配成功，就返回一个匹配成功的对象，否则返回 None。

【参数解释】

- pattern：匹配的正则表达式。
- string：要匹配的字符串。
- flags：标志位，用于控制正则表达式的匹配方式，如是否区分大小写、是否多行匹配等。

参数 flags 的可选值如下。

- re.I(re.IGNORECASE)：忽略大小写。
- re.M(MULTILINE)：多行模式，改变 '^' 和 '$' 的行为。
- re.S(DOTALL)：此模式下，'.' 的匹配不受限制，可匹配任何字符，包括换行符。
- re.L(LOCALE)：字符集本地化，为了支持多语言版本的字符集使用环境，比如转义符 \w。
- re.U(UNICODE)：使预定字符类 \w \W \b \B \s \S \d \D 取决于 unicode 定义的字符属性。

- re.X(VERBOSE)：详细模式。在这个模式下，正则表达式可以是多行的，忽略空白字符，并可以加入注释。

该函数匹配之后，得出一个 match 对象类型，如果要返回结果，那么可以使用 group() 或 groups() 匹配对象函数来获取匹配后的结果。例子如下：

```
import re
text = "This is the last one"
res = re.match('(.*) is (.*?) .*', text, re.M | re.I)
if res:
    print("res.group() : ", res.group())
    print("res.group(1) : ", res.group(1))
    print("res.group(2) : ", res.group(2))
    print("res.groups() : ", res.groups())
else:
    print("No match!!")
```

输出结果：

```
res.group() :  This is the last one
res.group(1) :  This
res.group(2) :  the
res.groups() :  ('This', 'the')
```

对于代码中的“(.*)”，“.”代表匹配任意字符，“*”代表匹配前一个字符 0 次或多次，两者结合匹配出“This”；在“is”后面的“(.*?) .*”代表获取 is 后面的全部数据，其中小括号匹配结果，“.*?”用于匹配一个单词“the”，而括号外的“.*”用于匹配任意字符，但不返回给匹配结果。如果对这部分较难理解，读者可以自行比较以下匹配结果：

```
res = re.match('(.*) is (.*?) (.*)', text, re.M | re.I)
res = re.match('(.*) is (.*) (.*)', text, re.M | re.I)
res = re.match('(.*) is (.*)', text, re.M | re.I)
res = re.match('(.*) is (.*?)', text, re.M | re.I)
```

（2）re.search(pattern, string, flags=0)，扫描整个字符串并返回第一次成功匹配的对象，如果匹配失败，就返回 None。

【参数解释】

- pattern：匹配的正则表达式。
- string：要匹配的字符串。
- flags：标志位，用于控制正则表达式的匹配方式，如是否区分大小写、是否多行匹配等，flags 可选值与 match 一样。

匹配结果跟 re.match 函数一样，使用 group() 和 groups() 方法来获取。

将 re.match 的例子改为 re.search，得出的结果是一致的。search 和 match 的使用方法相似，不过两者运行逻辑不同，两者的主要区别：re.match 只匹配字符串的开始，如果字符串开始不符合正则表达式，匹配就会失败，函数返回 None；而 re.search 匹配整个字符串，直到找到一个匹配的字符串，否则也返回 None。

（3）re.findall(pattern, string, flags=0)，获取字符串中所有匹配的字符串，并以列表的形式返回。列表中的元素有如下几种情况：

① 当正则表达式中含有多个圆括号时，返回的列表元素为多个字符串组成的元组，而且元组中的字符串个数与括号对数相同，并且字符串排放顺序跟括号出现的顺序一致，字符串的内容与每个括号内的正则表达式相对应。

② 当正则表达式中只带有一个圆括号时，返回的列表元素为字符串，并且该字符串的内容与括号中的正则表达式相对应。（注意：返回的列表（字符串）只是圆括号中的内容，不是整个正则表达式所匹配的内容。）

③ 当正则表达式中没有圆括号时，返回的列表中的字符串表示整个正则表达式匹配的内容。

【参数解释】

- pattern：匹配的正则表达式。
- string：要匹配的字符串。
- flags：标志位，用于控制正则表达式的匹配方式，如是否区分大小写、是否多行匹配等，flags 可选值与 match 一样。

匹配结果不需要使用 group() 和 groups() 方法来获取。例子如下：

```
import re
# 匹配字符串中所有含有 'oo' 字符的单词
# 当正则表达式中没有圆括号时，列表中的字符串表示整个正则表达式匹配的内容
find_value = re.findall('\w*oo\w*', 'woo this foo is too')
print(find_value)

# 获取字符串中所有的数字字符串
# 当正则表达式中只带有一个圆括号时，列表中的元素为字符串，并且该字符串的内容
与括号中的正则表达式相对应
find_value = re.findall('.*?(\d+).*?', 'adsd12343.jl34d5645fd789')
print(find_value)

# 提取字符串中所有有效的域名地址
# 正则表达式中有多个圆括号时，返回匹配成功的列表中的每一个元素都是由一次匹配
# 成功后，正则表达式中所有括号中匹配的内容组成的元组
add = 'https://www.net.com.edu//action=?asdfsd and other https://
www.baidu.com//a=b'
find_value = re.findall('((w{3}\.)(\w+\.)+(com|edu|cn|net))', add)
print(find_value)
```

输出结果：

```
['woo', 'foo', 'too']
['12343', '34', '5645', '789']
[('www.net.com.edu', 'www.', 'com.', 'edu'), ('www.baidu.com',
'www.', 'baidu.', 'com')]
```

（4）re.sub(pattern, repl, string, count=0, flags=0)，用于替换字符串中的匹配项，如果没有匹配的项，则返回没有匹配的字符串。

【参数解释】

- pattern：匹配的正则表达式。
- repl：用于替换的字符串。
- string：要被替换的字符串。
- count：替换的次数。
- flags：标志位，用于控制正则表达式的匹配方式，如是否区分大小写、是否多行匹配等，flags 可选值与 match 一样。

匹配结果不需要使用 group() 和 groups() 方法来获取。例子如下：

```
import re
# 将手机号的后 4 位替换成 0
replace_value = re.sub('\d{4}$', '0000', '13435423143')
print(replace_value)
# 将代码后面的注释信息去掉
replace_value = re.sub('#.*$', '', 'num = 0 #a number')
print(replace_value)
```

输出结果：

```
13435420000
num = 0
```

上述是正则处理函数的常用函数，除此之外，正则处理函数还有：

- re.split(pattern, string[, maxsplit])：用匹配 pattern 的子串来分割字符串。
- re.subn(pattern, repl, string[, count])：与 sub() 函数一样，只是返回结果是一个元组。
- re.escape(string)：把字符串里除了字母和数字以外的字符都加上反斜杆。
- re.finditer(pattern, string[, flags])：搜索字符串，按顺序返回每一个匹配结果的迭代器。

在学习正则表达式时，难点是如何根据字符串内容编写正确的正则表达式，元字符之间不同的组合会产生不同的结果，要熟练掌握正则表达式，必须熟知每个元字符的作用以及正则处理函数的使用方法。

8.3 Beautiful Soup 介绍及安装

Beautiful Soup 是一个可以从 HTML 或 XML 文件中提取数据的 Python 库。其功能简单而强大，容错能力高，文档相对完善，清晰易懂，具有三个特性：

（1）Beautiful Soup 提供了一些简单的方法和 Python 术语，用于检索和修改语法树：一个用于解析文档并提取相关信息的工具包。

（2）Beautiful Soup 自动将输入文档转换为 Unicode 编码，并将输出文档转化为 UTF-8 编码。不需要考虑编码，除非输入文档没有指出其编码并且 Beautiful Soup 无法自动检测到，这时需要指出原来的编码方式。

（3）Beautiful Soup 位于一些流行的 Python 解析器中，比如 lxml 和 html5lib 的上层，这允许使用不同的解析策略或者牺牲速度来换取灵活性。

Beautiful Soup 的安装涉及第三方的扩展，建议使用 pip 安装。

```
pip install beautifulsoup4
```

Beautiful Soup 支持 Python 标准库中的 HTML 解析器，还支持一些第三方的解析器，常用的解析器有 lxml 和 html5lib。lxml 的安装如下：

```
pip install lxml
```

lxml 是一个用来处理 XML 的第三方 Python 库，它的底层封装了由 C 语言编写的 libxml2 和 libxslt，并以简单、强大的 Python API 兼容并加强了著名的 ElementTree API。

另一个可供选择的解析器是纯 Python 实现的 html5lib，这是一个 Ruby 和 Python 用来解析 HTML 文档的类库，支持 HTML5 以及最大程度兼容桌面浏览器。使用 pip 安装 html5lib：

```
pip install html5lib
```

完成上述安装后，在 CMD（终端）下验证安装是否成功，打开 Python 交互式命令行（输入“Python”，按回车键即可），输入代码验证即可：

```
>>> import html5lib
>>> html5lib.__version__
'0.999999999'
>>> import lxml
>>> import bs4
>>> bs4.__version__
'4.6.0'
```

每种解析器都有自己的优缺点，对比如表 8-3 所示。

表8-3 各种解析器的比较

解析器	使用方法	优势	劣势
标准库	BeautifulSoup(html, "html.parser")	内置标准库，速度适中，文档容错能力强	Python3.2版本前的文档容错能力差
lxml HTML	BeautifulSoup(html, "lxml")	速度快，文档容错能力强	安装C语言库
lxml XML	BeautifulSoup(html, "xml")	速度快，唯一支持XML	安装C语言库
html5lib	BeautifulSoup(html, "html5lib")	容错性最强，可生成HTML5	运行慢，扩展差

在选择解析器的时候，要从实际出发，解析同一个 HTML，不同的解析器因为容错性不同会导致不同的结果，若得到的数据与实际存在差异，出现数据丢失和解析出来的数据与实际不相符，则有可能是解析器在解析 HTML 时出现错误，可选择不同的解析器对错误逐一排查。

8.4 Beautiful Soup 的使用

通过例子说明如何使用 Beautiful Soup，以 MySoup.html 文件内容为例：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title> Python</title>
</head>
<body>
<p id="python">
<a href="/index.html"> Python </a>BeautifulSoup 的使用
</p>
<p class="myclass">
<a href="http://www.baidu.com/">这是</a> 一个指向百度的页面的 URL。
</p>
</body>
</html>
```

在文件 MySoup.html 的同一目录创建 Soup_py.py 文件，Soup_py.py 的代码如下：

```
# 引入 BeautifulSoup
from bs4 import BeautifulSoup
# 读取 MySoup.html 文件
Open_file = open('MySoup.html', 'r', encoding='utf-8')
# 将 MySoup.html 的内容赋值给 Html_Content，并关闭文件
Html_Content = Open_file.read()
Open_file.close()
# 使用 html5lib 解释器解释 Html_Content 的内容
soup = BeautifulSoup(Html_Content, "html5lib")
# 输出 title
print('html title is ' + soup.title.getText())
# 查找第一个标签 p，并输出
find_p = soup.find('p', id="python")
print('the first <p> is ' + find_p.getText())
# 查找全部标签 p，并输出
find_all_p = soup.find_all('p')
for i, k in enumerate(find_all_p):
    print('the ' + str(i + 1) + ' p is ' + k.getText())
```

运行 Soup_py.py，结果如图 8-2 所示。

```
html title is  Python网络爬虫实战揭秘
the first <p> is
 Python BeautifulSoup的使用

the 1 p is
 Python BeautifulSoup的使用

the 2 p is
这是 一个指向百度的页面的链接。
```

图 8-2 BeautifulSoup 解析 HTML

代码运行时，先读取 HTML 文件的内容，将内容定义到一个 BeautifulSoup 对象中，并使用 html5lib 解析 HTML 内容，然后使用 Beautiful Soup 内置的方法找出标题的值和 <p> 的值。

前面的例子只是简单介绍了 Beautiful Soup 的基本用法，下面以 Python 的交互式命令行演示 Beautiful Soup 的更多用法（在 CMD（终端）中输入“Python”，按回车键可进入 Python 的交互式命令行）。

（1）查找全部标签，代码如下：

```
Html_content = """<html><head><title> Python</title></head>
<p class="title"><b>Beautiful Soup 的学习 </b></p>
<p class=” study” > 学习网址：http://blog.csdn.net/huangzhang_123
```

```
<a href="www.xxx.com" class="abc" id="try1">web 开发 </a>,
<a href=" www.ccc.com " class="bcd" id="try2"> 网络爬虫 </a> and
<a href=" www.aaa.com " class="efg" id="try3"> 人工智能 </a>;
</p>
<p class="other">...</p>"""

from bs4 import BeautifulSoup
soup = BeautifulSoup(Html_content, "html5lib")
# 以下是查找某标签的方法：
# 获取头部的信息，返回 <head></head> 之间的全部内容
soup.head
# 获取 title 的信息，返回 <title></title> 之间的全部内容
soup.title
# 这是一个获取 tag 的小窍门，可以在文档树的 tag 中多次调用这个方法。下面的代码可以获取 <body> 标签中的第一个 <b> 标签
# 也就是说，soup 不一定是整个 html 的内容，可以先定位某部分，然后用这个简洁的方式获取
# 返回 "<b>Beautiful Soup 的学习 </b>"
soup.body.b
# 直接指定标签类别，返回第一个标签的内容。返回 "<a href="www.xxx.com" class="abc"
# id = "try1">web 开发 </a>"
soup.a
# 获取第一个标签 a
soup.find_all('a')
#[<a href="www.xxx.com" class="abc" id="try1">web 开发 </a>,
#<a href=" www.ccc.com " class="bcd" id="try2"> 网络爬虫 </a>,
#<a href=" www.aaa.com " class="efg" id="try3"> 人工智能 </a>]
```

变量 Html_content 是一个 HTML 内容，其格式为字符串，Beautiful Soup 对 Html_content 生成对象 soup。数据获取是从 soup 对象获取，比如获取 head 和 title，可从 soup.head 和 soup.title 直接获取。想获取某个标签值，如 soup.a，返回的数据格式是<class 'bs4.element.Tag'>，这是 Beautiful Soup 的格式，代表第一个标签的全部内容，若想获取其标签在网页上显示的内容（去除 HTML 代码），则可通过以下方法：

① 通过 getText() 获取标签的值。例如，soup.a.getText() 返回的是“web 开发”。

② 通过 str() 方式转换为字符串。例如，str(soup.a) 返回的是“web 开发 ”，然后使用字符串截取获取的数据。

（2）获取某标签的属性值，在上述例子中，soup.a 可以获取第一个 HTML 的标签 a，如果想获取该标签里面的属性值，沿用上述变量 Html_content，实现代码如下：

```
soup = BeautifulSoup(Html_content, "html5lib")
print(soup.a['class'])
# 输出内容：'abc'
```

值得注意的是，在 HTML 中，class 属性可带有多个 CSS 样式，如果 HTML 的属性含有多个 CSS 样式，Beautiful Soup 会以列表的格式返回结果。例子如下：

```
soup = BeautifulSoup('<a href="www.xxx.com" class="abc bcd">web 开
发 </a>', "html5lib")
print(soup.a['class'])
# 输出内容：[ ‘abc’ , ‘bcd’ ]
```

（3）精准查找

上述例子只能返回第一个标签 a，如果想获取第 N 个标签 a 或者精确定位到某个标签，就只能使用其他方法实现。沿用上述变量 Html_content，实现精确定位标签 a，方法如下：

```
soup.find_all('a', id=" try3")
soup.find_all('a', class_="efg", id=" try3")
soup.find_all('a', href == re.compile("aaa"))
```

以上三种方式都可以定位到 人工智能 这个标签。

① 第一种是通过一个属性定位，只要是标签里具有的属性都可以定位到。

② 第二种在第一种的基础上增加了一种属性，也就是多个属性一起定位，这样更加精准。

③ 第三种是通过正则表达式进行模糊匹配，这个适合属性多变时使用。

在 Beautiful Soup 中，find() 和 find_all() 的使用方法一样。两者的区别在于：

① find_all() 返回的结果是包含一个或多个元素的列表；而 find() 方法返回的是第一个符合要求的结果，格式为字符串。

② 若 find_all() 没有找到目标，则返回空列表；而 find() 方法找不到目标时，返回 None。

（4）Beautiful Soup 支持大部分的 CSS 选择器。

CSS 样式定义由两部分组成，形式为：[code] 选择器 { 样式 } [/code]。

在 {} 之前的部分就是“选择器”。“选择器”指明了 {} 中“样式”的作用对象，也就是“样式”作用于网页中的哪些元素。

CSS 选择器主要是由前端的 CSS 编写的，这里简单介绍一下 Beautiful Soup 的 CSS 选择器的用法。

- 通过 id 查找：soup.select("#try3")。
- 通过 class 查找：soup.select(".efg")。
- 通过属性查找：soup.select('a[class="efg"]')。

上述三个方法也可以返回 人工智能 这个标签，与 find_all 实现的功能一样。

Beautiful Soup 在爬虫开发中担任着数据清洗的角色，掌握上述使用方法就能解决绝大部分的网站数据清洗问题。

8.5 本章小结

数据清洗是爬虫开发重要的一环，主要衔接了数据爬取和数据入库。常用的数据清洗方法有：字符串操作、正则表达式和第三方模块（库）。

常用数据清洗的字符串操作有截取、替换、查找和分割。

- 截取：字符串 [开始位置 : 结束位置 : 间隔位置]。
- 替换：字符串 .replace(' 被替换内容 ', ' 替换后内容 ')。
- 查找：字符串 .find(' 要查找的内容 ' [，开始位置，结束位置])。
- 分割：字符串 .split(' 分割符 ', 分割次数)。

正则表达式包含正则语法和正则处理函数。

- 正则语法：也称元字符，这类符号代表正则规则，通常表示一些不寻常的匹配操作，或者通过重复、修改匹配意义来影响正则模式的其他部分。
- 正则处理函数：Python 的正则模块是 re，该模块含有多种正则处理函数，功能函数包括：

（1）re.match(pattern, string, flags=0)

（2）re.search(pattern, string, flags=0)

（3）re.findall(pattern, string, flags=0)

（4）re.sub(pattern, repl, string, count=0, flags=0)

（5）re.split(pattern, string[, maxsplit])

（6）re.subn(pattern, repl, string[, count])

（7）re.escape(string)

（8）re.finditer(pattern, string[, flags])

Beautiful Soup 是一个可以从 HTML 或 XML 文件中提取数据的 Python 库。常用的数据清洗函数如下。

- 查找全部标签：soup.a，返回第一个标签 a。
- 获取定位元素（标签）的值：soup.a.getText()，获取第一个标签 a 的值。
- 获取标签属性：soup.a['href']，获取整个 HTML 第一个标签 a 的 href 属性值。
- 精准查找：find_all() 和 find()。
  - 属性定位：soup.find_all('a', id=" try3")。
  - 多属性定位：soup.find_all('a', class_="efg", id=" try3")。
  - 正则表达式模糊匹配：soup.find_all('a', href == re.compile("aaa"))。
- CSS 选择器。
  - 通过 id 查找：soup.select("#try3")。
  - 通过 class 查找：soup.select(".efg")。
  - 通过属性查找：soup.select('a[class="efg"]')。

第 9 章 文档数据存储

9.1 CSV 数据写入和读取

常用的数据存储介质有文件、关系式数据库和非关系式数据库。文本文档存储适用于具有时效性的数据，如股市行情、商品信息和排行榜信息等，这类数据具有动态变化性质，非特殊要求下，建议存放文件。

Python 标准库自带 CSV 模块，不用自行安装。数据写入 CSV 的代码如下：

```
import csv
# 若存在文件，则打开 csv 文件；若不存在，则新建文件
# 若不设置 newline=''，则每行数据会隔一行空白行
csvfile = open('csv_test.csv', 'w', newline='')
# 将文件加载到 csv 对象中
```

```
writer = csv.writer(csvfile)
# 写入一行数据
writer.writerow(['姓名', '年龄', '电话'])
# 多行数据写入
data = [
    ('小P', '18', '138001380000'),
    ('小Y', '22', '138001380000')
]
writer.writerows(data)
# 关闭 csv 对象
csvfile.close()
```

写入 CSV 时使用 open 函数打开文件，open 函数最好设置参数“newline”为空，否则每次写入一行数据，数据之间就会空出一行空白行。将打开的文件对象加载到 CSV 对象中，写入数据分为单行写入和多行写入，对应函数分别是 writerow 和 writerows。

读取 CSV 文件，读取函数有 reader 和 DictReader，两者都是接收一个可迭代的对象，返回一个生成器。reader 函数是将一行数据以列表形式返回；DictReader 函数返回的是一个字典，字典的值是单元格的值，而字典的键则是这个单元格的标题（列头）。代码如下：

```
import csv
csvfile = open('csv_test.csv', 'r')
# 以列表形式输出
reader = csv.reader(csvfile)
# 以字典形式输出，第一行作为字典的键
# reader = csv.DictReader(csvfile)
rows = [row for row in reader]
print(rows)
```

上述代码用于获取全部数据，如果要获取某行数据，就可以循环全部数据，再对每行数据做一个判断，判断是否符合筛选条件，代码如下：

```
import csv
csvfile = open('csv_test.csv', 'r')
# 以列表形式输出
reader = csv.reader(csvfile)
```

```
for row in reader:
    if '小P' in row:
        print(row)
# 以字典形式输出，第一行作为字典的键
# reader = csv.DictReader(csvfile)
# for row in reader:
#     if row['姓名']== '小P':
#         print(row)
```

要获取某行数据，使用不同函数会有不同的判断方式，reader 函数返回的是列表，DictReader 返回的是字典，要根据某个值判断筛选，所采用的方法也不一样。CSV 的存储相对较为简单，而且实用性比较强。

9.2 Excel 数据写入和读取

Python 操作的 Excel 库有 xlrd、xlwt、pyExcelerator 和 openpyxl。其中，pyExcelerator 只支持 2003 版本，openpyxl 只支持 2007 版本，xlrd 支持 Excel 任何版本的读取，xlwt 支持 Excel 任何版本的写入。

为了版本的兼容性，大多数开发人员选择使用 xlrd 和 xlwt 操作 Excel。xlrd 和 xlwt 的安装如下：

```
pip install xlrd
pip install xlwt
```

完成安装后，在 Python 交互式命令行输入验证代码：

```
>>> import xlwt
>>> import xlrd
>>> xlrd.__VERSION__
'1.1.0'
>>> xlwt.__VERSION__
'1.3.0'
```

Excel 的写入相对比 CVS 复杂，Excel 可以实现设置数据格式、合并单元格、设置公式和插入图片等功能。使用 xlwt 实现上述功能的代码如下：

```
import xlwt
# 新建一个 Excel 文件
wb = xlwt.Workbook()
# 新建一个 Sheet
ws = wb.add_sheet('Python', cell_overwrite_ok=True)
# 定义字体对齐方式对象
alignment = xlwt.Alignment()
# 设置水平方向
# HORZ_GENERAL, HORZ_LEFT, HORZ_CENTER, HORZ_RIGHT, HORZ_FILLED
# HORZ_JUSTIFIED, HORZ_CENTER_ACROSS_SEL, HORZ_DISTRIBUTED
alignment.horz = xlwt.Alignment.HORZ_CENTER
# 设置垂直方向
# VERT_TOP, VERT_CENTER, VERT_BOTTOM, VERT_JUSTIFIED, VERT_
DISTRIBUTED
alignment.vert = xlwt.Alignment.VERT_CENTER
# 定义格式对象
style = xlwt.XFStyle()
style.alignment = alignment
# 合并单元格 write_merge( 开始行 , 结束行 , 开始列 , 结束列 , 内容 , 格式 )
ws.write_merge(0, 0, 0, 5, 'Python 网络爬虫 ', style)

# 写入数据 wb.write( 行 , 列 , 内容 )
for i in range(2, 7):
    for k in range(5):
        ws.write(i, k, i+k)
    # Excel 公式 xlwt.Formula
    ws.write(i, 5, xlwt.Formula('SUM(A'+str(i+1)+':E'+str
(i+1)+')'))

# 插入图片，insert_bitmap(img, x, y, x1, y1, scale_x=0.8, scale_
y=1)
# 图片格式必须为 bmp
# x 表示行数，y 表示列数
# x1 表示相对原来位置向下偏移的像素
# y1 表示相对原来位置向右偏移的像素
# scale_x、scale_y 缩放比例
ws.insert_bitmap('E:\\test.bmp', 9, 1, 2, 2, scale_x=0.3, scale_
y=0.3)
```

```
# 保存文件
wb.save('file.xls')
```

代码依次实现的功能如下。

- 设置字体水平垂直居中：该功能实现共分为两步，第一步是定义 xlwt.Alignment() 对象，分别设置其水平方向和垂直方向的属性；第二步是定义 xlwt.XFStyle() 对象，将设置好的 Alignment() 对象赋予 XFStyle() 对象。在写入数据的时候，XFStyle() 对象作为 write_merge() 方法的参数。
- 合并单元格：主要由 write_merge(开始行 , 结束行 , 开始列 , 结束列 , 内容 , 格式) 方法实现。
- 生成表格并计算每行总和：通过嵌套循环生成 5 行 6 列的表格，第 1 到第 5 列的数据写入由 write() 方法实现；第 6 列数据是累计求和，由 Excel 自带公式实现。
- 插入图片：图片插入是由 insert_bitmap(img, x, y, x1, y1, scale_x=0.8, scale_y=1) 实现的，图片格式必须为 bmp，否则无法插入并提示错误。

把数据写入 Excel 的整体思路如下：

（1）xlwt 创建生成临时 Excel 对象。

（2）添加 WorkSheets 对象。

（3）单元格的位置由行列索引决定，索引从 0 开始。

（4）数据写入主要由 write_merge() 和 write() 实现，两者分别是合并单元格再写入和单元格写入。

（5）设置数据格式是在写入（write_merge() 和 write()）的数据中传入参数 style。

运行程序，结果如图 9-1 所示。

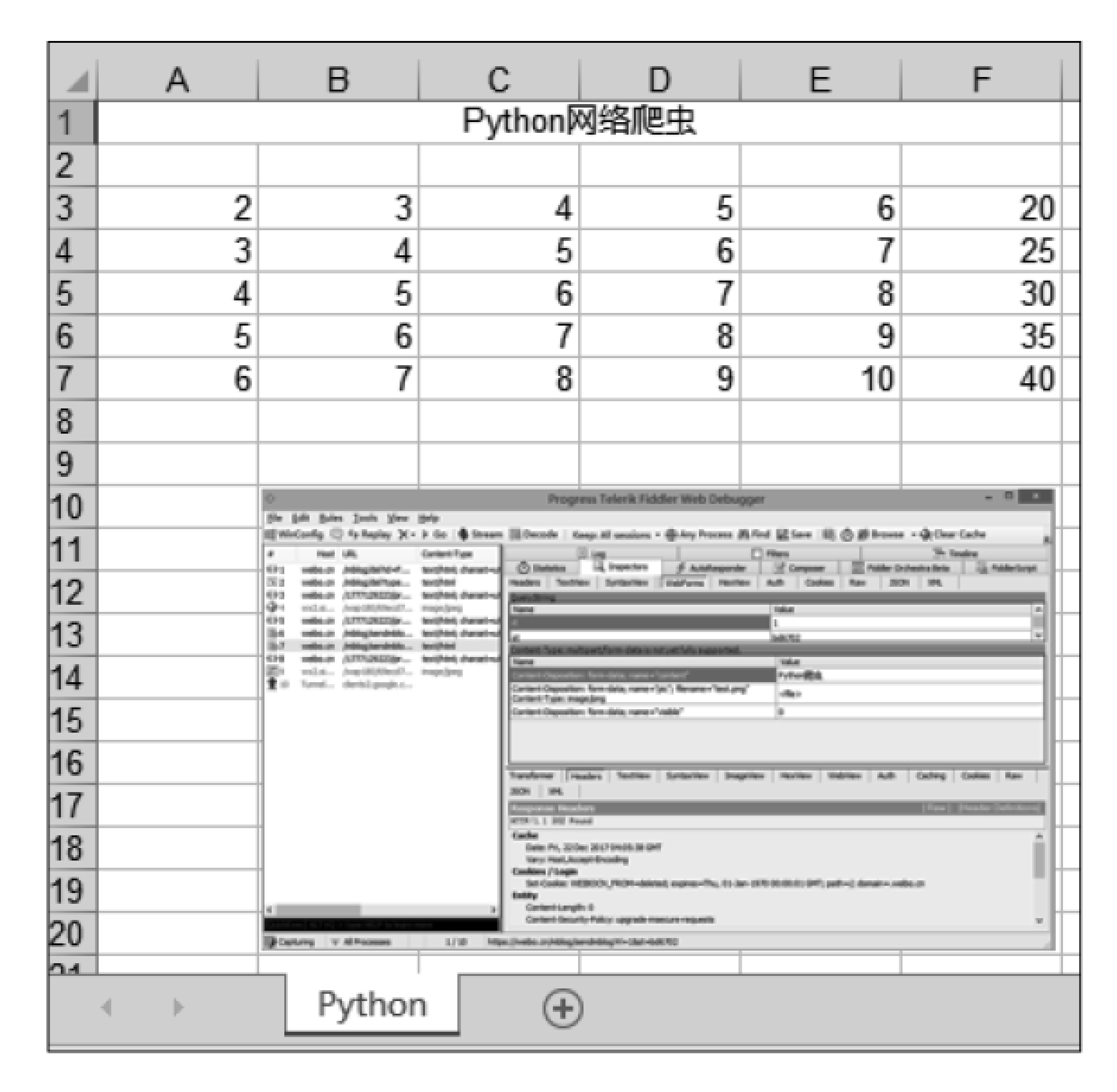


图 9-1　在 Excel 中写入数据

除此之外，xlwt 还可以设置单元格背景颜色、添加单元格边框、设置单元格高宽度、设置字体颜色和数据类型等，由于篇幅较大，本书就不一一讲解了。

接着读取 Excel 数据，由 xlrd 模块实现，我们以上述已生成的 Excel 为读取目标，代码如下：

```
import xlrd
wb = xlrd.open_workbook('file.xls')
# 获取 Sheets 总数
ws_count = wb.nsheets
print('Sheets 总数：', ws_count)
# 通过索引顺序获取 Sheets
# ws = wb.sheets()[0]
# ws = wb.sheet_by_index(0)
# 通过 Sheets 名获取 Sheets
ws = wb.sheet_by_name('Python')
# 获取整行的值（以列表返回内容）
row_value = ws.row_values(3)
print('第 4 行数据：', row_value)
# 获取整列的值（以列表返回内容）
row_col = ws.col_values(3)
```

```
print('D 列数据: ', row_col)

# 获得所有行列
nrows = ws.nrows
ncols = ws.ncols
print(' 总行数: ', nrows, ', 总列数: ', ncols)

# 获取某个单元格内容 cell( 行 , 列 )
cell_F3 = ws.cell(2, 5).value
print('F3 内容: ', cell_F3)

# 使用行列索引获取某个单元格内容
row_F3 = ws.row(2)[5].value
col_F3 = ws.col(5)[2].value
print('F3 内容: ', row_F3, 'F3 内容: ', col_F3)
```

运行程序，结果如图 9-2 所示。

```
Sheets总数:  1
第4行数据:  [3.0, 4.0, 5.0, 6.0, 7.0, 25.0]
D列数据:  ['', '', 5.0, 6.0, 7.0, 8.0, 9.0]
总行数:  7 , 总列数:  6
F3内容:  20.0
F3内容:  20.0 F3内容:  20.0
```

图 9-2 从 Excel 中读取数据

读取 Excel 的数据思路大致如下：

（1）xlrd 生成 Workbook 对象，并指向 Excel 文件。

（2）选择 Workbook 里某个 WorkSheets 对象。

（3）获取 WorkSheets 里数据已占用的总行数和总列数（某个单元格数据）。

（4）循环总行数和总列数，读取每一个单元格的数据。

9.3 Word 数据写入和读取

将数据存储在 Word 文档中，一般以文章、新闻报道和小说这类文字内容较长的数据为主。

Python 读写 Word 需要第三方库扩展支持，使用 pip 安装：

```
pip install python-docx
```

模块安装后，验证模块是否安装成功，在 Python 交互式命令行输入验证代码：

```
>>> import docx
>>> docx.__version__
'0.8.6'
```

下面通过例子来讲述如何将数据写入 Word 文档，代码如下：

```
# 数据写入
from docx import Document
from docx.shared import Inches
# 创建对象
document = Document()
# 添加标题，其中“0”代表标题类型，共有 4 种类型，具体可在 Word 的“开始”→“样式”中查看
document.add_heading('Python 爬虫 ', 0)
# 添加正文内容并设置部分内容格式
p = document.add_paragraph('Python 爬虫开发 -')
# 设置内容加粗
p.runs[0].bold = True
# 添加内容并加粗
p.add_run(' 数据存储 -').bold = True
# 添加内容
p.add_run('Word-')
# 添加内容并设置字体斜体
p.add_run(' 存储实例。').italic = True
# 添加正文，设置“样式”→“明显引用”
document.add_paragraph(' 样式 '-' 明显引用 ', style='IntenseQuote')
# 添加正文，设置“项目符号”
```

```
document.add_paragraph(
    '项目符号1', style='ListBullet'
)
document.add_paragraph(
    '项目符号2', style='ListNumber'
)
# 添加图片
document.add_picture('test.png', width=Inches(1.25))
# 添加表格
table = document.add_table(rows=1, cols=3)
hdr_cells = table.rows[0].cells
hdr_cells[0].text = 'Qty'
hdr_cells[1].text = 'Id'
hdr_cells[2].text = 'Desc'
for item in range(2):
    row_cells = table.add_row().cells
    row_cells[0].text = 'a'
    row_cells[1].text = 'b'
    row_cells[2].text = 'c'
# 保存文件
document.add_page_break()
document.save('test.docx')
```

在 Word 中写入数据的整体思路如下：

（1）创建生成临时 Word 对象。

（2）分别使用 add_paragraph() 和 add_heading() 对 Word 对象添加标题和正文内容。

（3）如果想设置正文内容的字体加粗和斜体等，可以将正文内容 p 对象的属性 runs[0].bold 和 add_run('XX').italic 设置为 True。

（4）如果要插入图片和添加表格，可以在 Word 对象中使用方法 add_picture() 和 add_table()。

（5）完成数据写入，需要将 Word 对象保存成 Word 文件。

读取 Word 数据比写入数据相对简单，因为不用设置内容格式，直接获取数据即可。实现代码如下：

```
# 数据读取
import docx
def readDocx(docName):
    fullText = []
    doc = docx.Document(docName)
    # 读取全部内容
    paras = doc.paragraphs
    # 将每行数据存入列表
    for p in paras:
        fullText.append(p.text)
        # 将列表数据转换成字符串
    return '\n'.join(fullText)
print(readDocx('test.docx'))
```

在 Word 中读取数据的整体思路如下：

（1）生成 Word 对象，并指向 Word 文件。

（2）使用 paragraphs() 获取 Word 对象全部内容。

（3）循环 paragraphs 对象，获取每行数据并写入列表。

（4）将列表转换为字符串，每个列表元素使用换行符连接，转换后数据的段落布局与 Word 文档相似。

9.4 本章小结

写入和读取 CSV、Excel 和 Word 中的数据是编写爬虫程序的重要内容，存入 CSV、Excel 和 Word 中的数据一般具体动态变化性质，有一定的时效性，适用于股市行情、商品信息、新闻报道和排行榜信息等。本章主要讲解了 CSV、Excel 和 Word 中数据的写入和读取方法，要点如下：

CSV 写入数据的整体思路：

（1）open 函数打开 CSV 文件，模式为 w（一般设置 newline=''），生成 file 对象。

（2）CSV 模块的 writer() 方法加载对象 file。

（3）使用 writerow()（writerows()）写入一行（多行）数据。

CSV 读取数据的整体思路：

（1）open 函数打开 CSV 文件，模式为 r，生成 file 对象。

（2）CSV 模块的 reader() 方法加载对象 file。

（3）使用 reader（DictReader）读取数据。

（4）reader 和 DictReader 区别：两者是接收一个可迭代的对象，返回一个生成器，reader 函数是将一行数据以列表形式返回；DictReader 函数返回的是一个字典，字典的值是单元格的值，而字典的键则是这个单元格的标题（即列头）

Excel 写入数据的整体思路：

（1）xlwt 创建生成临时 Excel 对象。

（2）添加 WorkSheets 对象。

（3）单元格的位置由行列索引决定，索引从 0 开始。

（4）数据写入主要由 write_merge() 和 write() 实现，两者分别是合并单元格再写入和单元格写入。

（5）设置数据格式是在写入（write_merge() 和 write()）数据传入参数 style。

Excel 读取数据的整体思路：

（1）xlrd 生成 Workbook 对象，并指向 Excel 文件。

（2）选择 Workbook 里某个 WorkSheets 对象。

（3）获取 WorkSheets 里数据已占用的总行数和总列数（某个单元格数据）。

（4）循环总行数和总列数，读取每一个单元格的数据。

Word 写入数据的整体思路：

（1）创建生成临时 Word 对象并使用以下方法添加内容：

（2）add_heading() 添加标题。

（3）add_paragraph() 添加正文内容。

（4）add_picture() 插入图片。

（5）add_table() 添加表格。

Word 读取数据的整体思路：

（1）生成 Word 对象，并指向 Word 文件。

（2）paragraphs() 获取 Word 对象全部内容。

（3）循环 paragraphs 对象，获取每行数据并写入列表。

（4）将列表转换为字符串，每个列表元素使用换行符连接，转换后数据的段落布局与 Word 文档相似。

第 10 章

ORM 框架

10.1 SQLAlchemy 介绍

关系数据库有 MySQL、Oracle、SQL Server、SQLite 和 PostgreSQL，操作数据库大致分为以下两种方式：

（1）直接使用数据库接口连接。在 Python 的关系数据库连接模块中，分别有 pymysql、cx_Oracle、pymssql、sqlite3 和 psycopg2。通常，这类数据库的操作步骤都是连接数据库、执行 SQL 语句、提交事务、关闭数据库连接。每次操作都需要 Open/Close Connection，如此频繁的操作对于整个系统无疑就成了一种浪费。对于一个企业级的开发来说，这无疑是不科学的开发方式。

（2）通过 ORM 框架来操作数据库。对象 - 关系映射（Object/Relation Mapping，ORM）是随着面向对象软件开发方法的发展而产生的。面向对象的开发方法是当今企

业级应用开发环境中的主流开发方法，关系数据库是企业级应用环境中永久存放数据的主流数据存储系统。对象和关系数据是业务实体的两种表现形式，业务实体在内存中表现为对象，在数据库中表现为关系数据。内存中的对象之间存在关联和继承关系，而在数据库中，关系数据无法直接表达多对多关联和继承关系。因此，对象 - 关系映射系统一般以中间件的形式存在，主要实现程序对象到关系数据库数据的映射。

在实际工作中，企业级开发都是使用 ORM 框架来实现数据库持久化操作的，所以作为一个开发人员，很有必要学习 ORM 框架，常用的 ORM 框架模块有 SQLObject、Stom、Django 的 ORM、peewee 和 SQLAlchemy。

本节主要讲述 Python 的 ORM 框架——SQLAlchemy。SQLAlchemy 是 Python 编程语言下的一款开源软件，提供 SQL 工具包及对象关系映射工具，使用 MIT 许可证发行。

SQLAlchemy 采用简单的 Python 语言，为高效和高性能的数据库访问设计，实现了完整的企业级持久模型。SQLAlchemy 的理念是，SQL 数据库的量级和性能重要于对象集合，而对象集合的抽象又重要于表和行。因此，SQLAlchmey 采用类似 Java 里 Hibernate 的数据映射模型，而不是其他 ORM 框架采用的 Active Record 模型。不过，Elixir 和 declarative 等可选插件可以让用户使用声明语法。

SQLAlchemy 首次发行于 2006 年 2 月，是 Python 社区中被广泛使用的 ORM 工具之一，不亚于 Django 的 ORM 框架。

SQLAlchemy 在构建于 WSGI 规范的下一代 Python Web 框架中得到了广泛应用，是由 Mike Bayer 及其开发团队开发的一个单独的项目。使用 SQLAlchemy 等独立 ORM 的一个优势就是允许开发人员首先考虑数据模型，并能决定稍后可视化数据的方式（采用命令行工具、Web 框架还是 GUI 框架）。这与先决定使用 Web 框架或 GUI 框架，再决定如何在框架允许的范围内使用数据模型的开发方法极为不同。

SQLAlchemy 的一个目标是提供能兼容众多数据库（如 SQLite、MySQL、Postgres、Oracle、MS-SQL、SQLServer 和 Firebird）的企业级持久性模型。

10.2 安装 SQLAlchemy

安装 SQLAlchemy 时，建议直接使用 pip 安装。

```
pip install SQLAlchemy
```

除了通过 pip 安装外，也可以在 www.lfd.uci.edu/~gohlke/pythonlibs/#sqlalchemy 下载 SQLAlchemy 版本的 whl 文件，whl 文件可使用 pip 安装，在 CMD（终端）中切换到 whl 文件所在路径，输入安装指令：

```
pip install SQLAlchemy-1.1.14-cp35-cp35m-win_amd64.whl
```

使用 SQLAlchemy 连接数据库实质上还是通过数据库接口实现连接，安装 SQLAlchemy 后还需要安装对应数据库的接口模块，下面以 MySQL 为例安装 pymysql 模块：

```
pip install pymysql
```

完成安装后，打开 CMD 窗口，通过导入模块测试是否安装成功：

```
>>> import sqlalchemy
>>> sqlalchemy.__version__
'1.1.12'
>>> import pymysql
>>> pymysql.__version__
'0.7.11.None'
```

10.3 连接数据库

在使用 SQLAlchemy 连接数据库之前，先简单介绍一下数据库系统环境，数据库系统版本信息如图 10-1 所示。

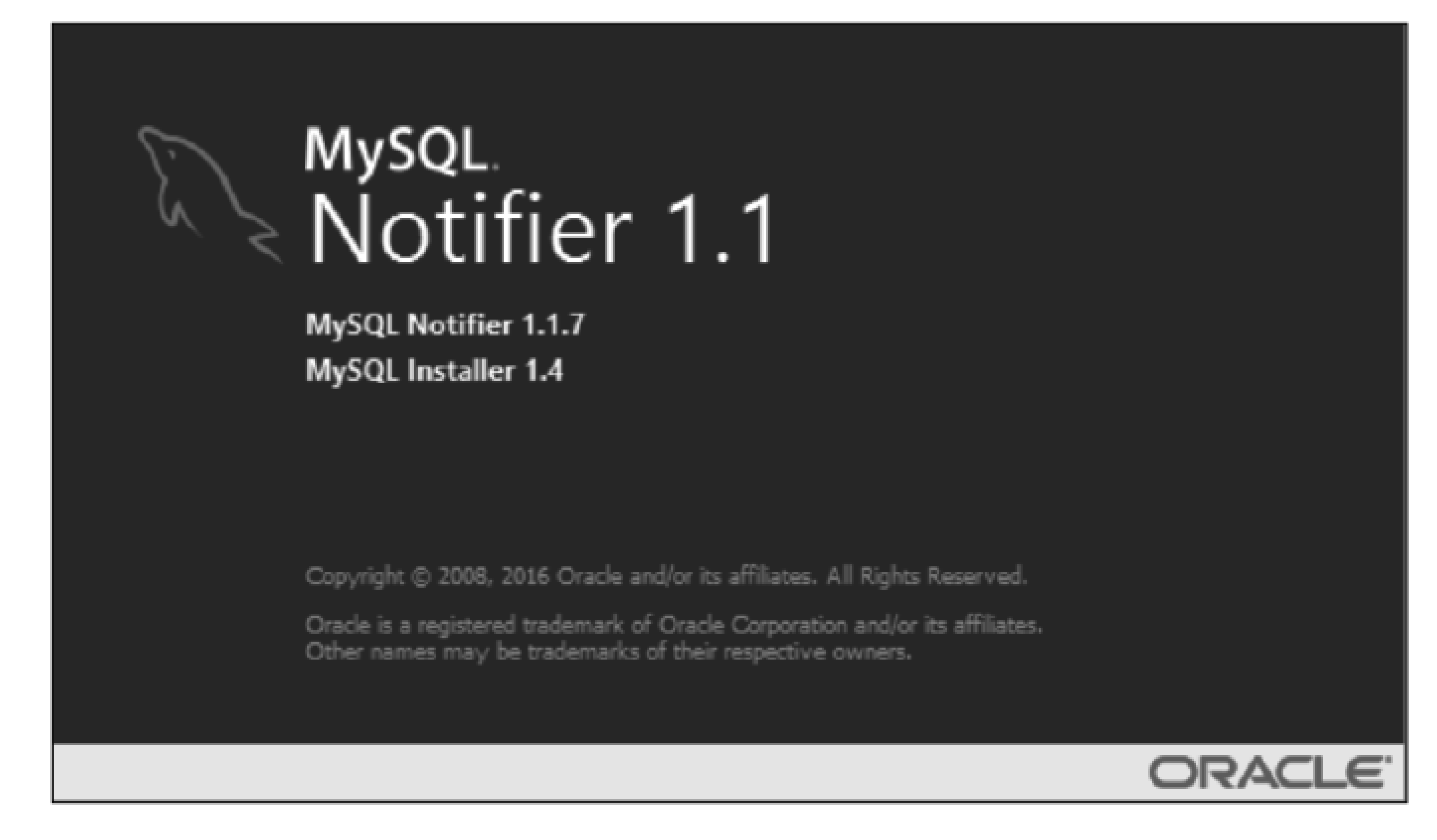


图 10-1 MySQL 信息

使用的数据库是本地数据库，端口是默认端口 3306，是通过 MySQL 工作台创建并命名为 test 的数据库，如图 10-2 所示。

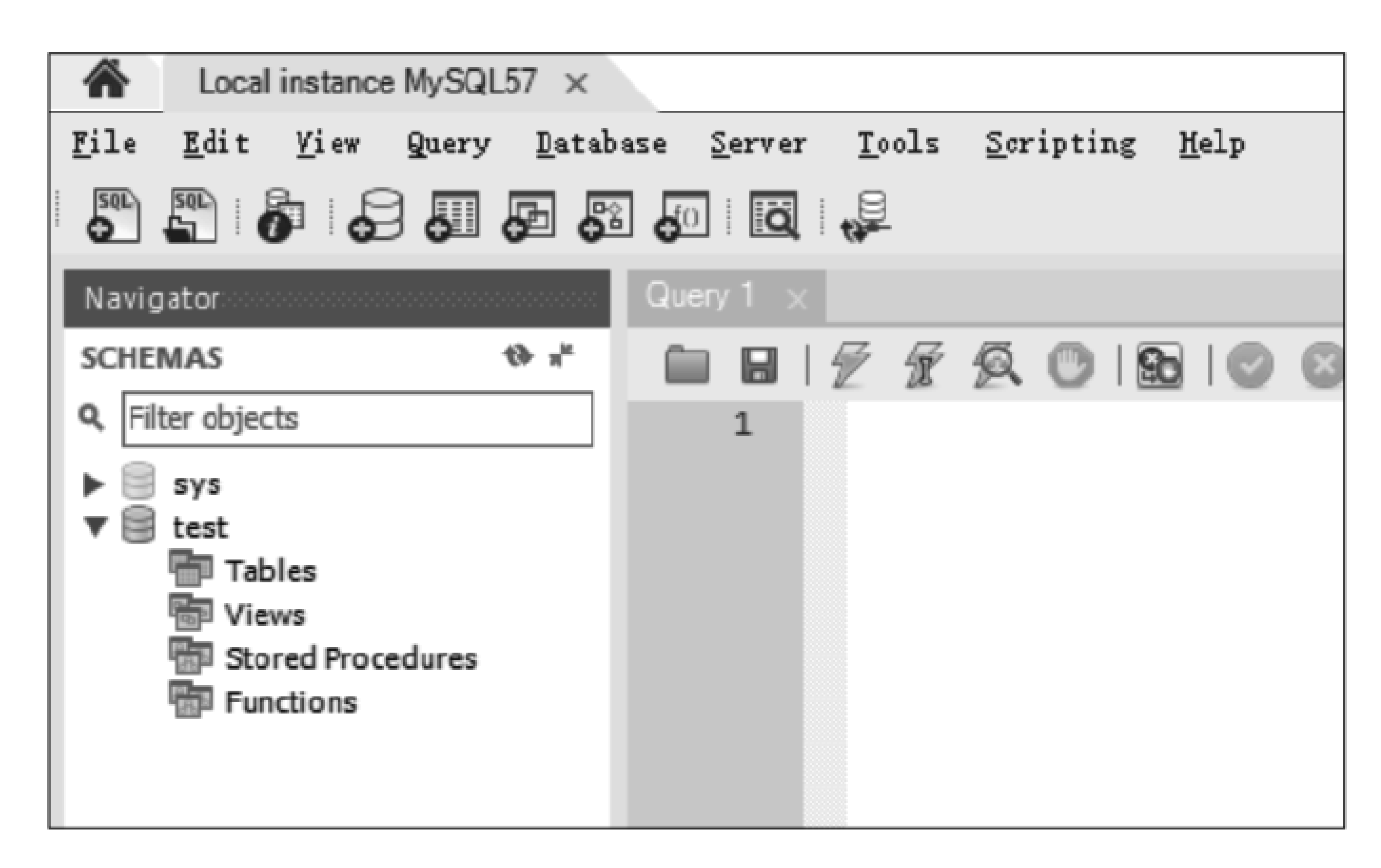


图 10-2 数据库信息

SQLAlchemy 连接数据库使用数据库连接池技术，原理是在系统初始化的时候，将数据库连接作为对象存储在内存中，当用户需要访问数据库时，并非建立一个新的连接，而是从连接池中取出一个已建立的空闲连接对象。使用完毕后，用户也并非将连接关闭，而是将连接放回连接池中，以供下一个请求访问使用。而连接的建立、断开都由连接池自身来管理。同时，还可以通过设置连接池的参数来控制连接池中的初始连接数、连接的上下限数以及每个连接的最大使用次数、最大空闲时间等。也可以通过其自身的管理机制来监视数据库连接的数量、使用情况等。

通过了解 SQLAlchemy 的原理有利于理解 SQLAlchemy 连接数据库的代码，代码如下：

```
from sqlalchemy import create_engine
engine=create_engine("mysql+pymysql://root:110@localhost:3306/
test?charset=utf8",echo=True)
```

导入 SQLAlchemy 的 create_engine 模块，设置数据库指令和参数后可实现连接，上述代码是常用的连接方式。create_engine 的参数设置说明如下。

- mysql+pymysql://root:110@localhost:3306/test：mysql 指明数据库系统类型，pymysql 是连接数据库接口的模块，root 是数据库系统用户名，110 是数据库系统密码，localhost:3306 是本地的数据库系统和数据库端口，test 是数据库名称。

- echo=True：用于显示 SQLAlchemy 在操作数据库时所执行的 SQL 语句情况，相当于一个监视器，可以清楚知道执行情况，如果设置为 False，就可以关闭。
- pool_size：设置连接数，默认设置 5 个连接数，连接数可以根据实际情况进行调整，在一般的爬虫开发中，使用默认值已足够。
- max_overflow：默认连接数为 10。当超出最大连接数后，如果超出的连接数在 max_overflow 设置的访问内，超出的部分还可以继续连接访问，在使用过后，这部分连接不放在 pool（连接池）中，而是被真正关闭。
- pool_recycle：连接重置周期，默认为 -1, 推荐设置为 7200, 即如果连接已空闲 7200 秒，就自动重新获取 , 以防止 connection 被关闭。
- pool_timeout：连接超时时间，默认为 30 秒，超过时间的连接都会连接失败。
- ?charset=utf8：对数据库进行编码设置，能对数据库进行中文读写，如果不设置，在进行数据添加、修改和更新等时，就会提示编码错误。

完整的连接数据库代码如下：

```
    from sqlalchemy import create_engine
    engine=create_engine("mysql+pymysql://root:110@localhost:3306/
test",echo=True,pool_size=5,
    max_overflow=4, pool_recycle=7200, pool_timeout=30)
```

上述代码只是给出连接 MySQL 的语句，其他数据的连接如表 10-1 所示。

表10-1　主流数据库连接方式

数据库	连接字符串
Microsoft SQL Server	mssql+pymssql://username:password@ip:port/dbname
MySQL	mysql+pymysql://username:password@ip:port/dbname
Oracle	cx_Oracle://username:password@ip:port/dbname
PostgreSQL	postgresql://username:password@ip:port/dbname
SQLite	sqlite://file_path

10.4 创建数据表

完成数据库的连接后，可以通过 SQLAlchemy 对数据表进行创建和删除，由图 10-2 可知，test 数据库是没有数据表的，使用 SQLAlchemy 创建数据表，代码如下：

```
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, String, DateTime
Base = declarative_base()

class mytable(Base):
    # 表名
    __tablename__ = 'mytable'
    # 字段，属性
    id = Column(Integer, primary_key=True)
    name = Column(String(50), unique=True)
    age = Column(Integer)
    birth = Column(DateTime)
    class_name = Column(String(50))
# 创建数据表
Base.metadata.create_all(engine)
```

引入 declarative_base 模块，生成其对象 Base，再创建一个类 mytable。一般情况下，数据表名和类名是一致的，__tablename__ 用于定义数据表的名称，可忽略，忽略时默认类名为数据表名。然后创建字段 id、name、age、birth、class_name。最后使用 Base.metadata.create_all(engine) 在数据库中创建对应的数据表。

上述是比较常见的创建数据表的方法之一，还有一种创建方法类似 SQL 语句的创建方法：

```
from sqlalchemy import Column, MetaData, ForeignKey, Table
from sqlalchemy.dialects.mysql import (INTEGER, CHAR)
meta = MetaData()
myclass = Table('myclass', meta,
                Column('id', INTEGER, primary_key=True),
                Column('name', CHAR(50), ForeignKey(mytable.name)),
                Column('class_name', CHAR(50))
                )
# 创建数据表
myclass.create(bind=engine)
```

此创建方法与前面介绍的创建数据表的方法大有不同，代码比较偏向于 SQL 创建数据表的语法，两者引入的模块也各不相同，导致在创建数据表的时候，创建语法

也不一致。不过两者实现的功能是一样的，读者可以根据自己的爱好进行选择。一般情况下，前者较有优势，在数据表已经存在的情况下，前者再创建数据表不会报错，后者就会提示已存在数据表的错误信息。

若要删除数据表，则可用以下代码：

```
# 先删除 myclass，后删除 mytable
myclass.drop(bind=engine)
Base.metadata.drop_all(engine)
```

在删除数据表的时候，一定要先删除设有外键的数据表，也就是先删除 myclass 后才能删除 mytable，两者之间涉及外键，这是在数据库中删除数据表的规则。

以下是完整的代码：

```
# 连接数据库
from sqlalchemy import create_engine
engine = create_engine(
    "mysql+pymysql://root:1990@localhost:3306/test?charset=utf8",
    echo=True)

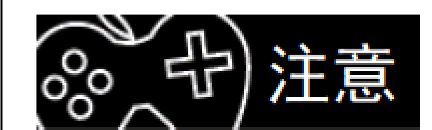
# 创建数据表方法一
from sqlalchemy import Column, Integer, String, DateTime
from sqlalchemy.ext.declarative import declarative_base
Base = declarative_base()

class mytable(Base):
    # 表名
    __tablename__ = 'mytable'
    # 字段，属性
    id = Column(Integer, primary_key=True)
    name = Column(String(50), unique=True)
    age = Column(Integer)
    birth = Column(DateTime)
    class_name = Column(String(50))

Base.metadata.create_all(engine)
```

```
# 创建数据表方法二
from sqlalchemy import Column, MetaData, ForeignKey, Table
from sqlalchemy.dialects.mysql import (INTEGER, CHAR)
meta = MetaData()
myclass = Table('myclass', meta,
                Column('id', INTEGER, primary_key=True),
                Column('name', CHAR(50), ForeignKey(mytable.name)),
                Column('class_name', CHAR(50))
                )
myclass.create(bind=engine)

# 删除数据表
myclass.drop(bind=engine)
Base.metadata.drop_all(engine)
```



无论数据表是否已经创建，在使用 SQLAlchemy 时一定要对数据表的属性、字段进行类定义。也就是说，无论通过什么方式创建数据表，在使用 SQLAlchemy 的时候，第一步是创建数据库连接，第二步是定义类来映射数据表，类的属性映射数据表的字段。

10.5 添加数据

完成数据表的创建后，下一步对数据表的数据进行操作。首先创建一个会话对象，用于执行 SQL 语句，代码如下：

```
from sqlalchemy.orm import sessionmaker
DBSession = sessionmaker(bind=engine)
session = DBSession()
```

引入 sessionmaker 模块，指明绑定已连接数据库的 engine 对象，生成会话对象 session，该对象用于数据库的增、删、改、查。

一般来说，常用的数据库操作是增、改、查，SQLAlchemy 对这类操作有自身的语法支持。对 10.4 节中创建的数据表添加数据，代码如下：

```
    new_data = mytable(name='Li Lei',age=10,birth='2017-10-01',
class_name='一年级一班')
    session.add(new_data)
    session.commit()
    session.close()
```

要使用 SQLAlchemy 添加数据，必须已经定义 mytable 对象，mytable 是映射数据库里面的 mytable 数据表。然后设置类属性（字段）对应的添加值，将数据绑定在 session 会话中，最后通过 session.commit() 来提交到数据中，就完成对数据库的数据添加了。session.close() 用于关闭会话，关闭会话不是必要规定，不过为了形成良好的编码规范，最好添加上。



如果关闭会话放在 session.commit() 之前，这个添加语句就是无效的，因为当前的 session 已经被关闭和销毁。所以在使用 session.close() 时，要注意编写的位置。

通过 MySQL 工作台可以看到数据表中已成功添加一条数据，如图 10-3 所示。

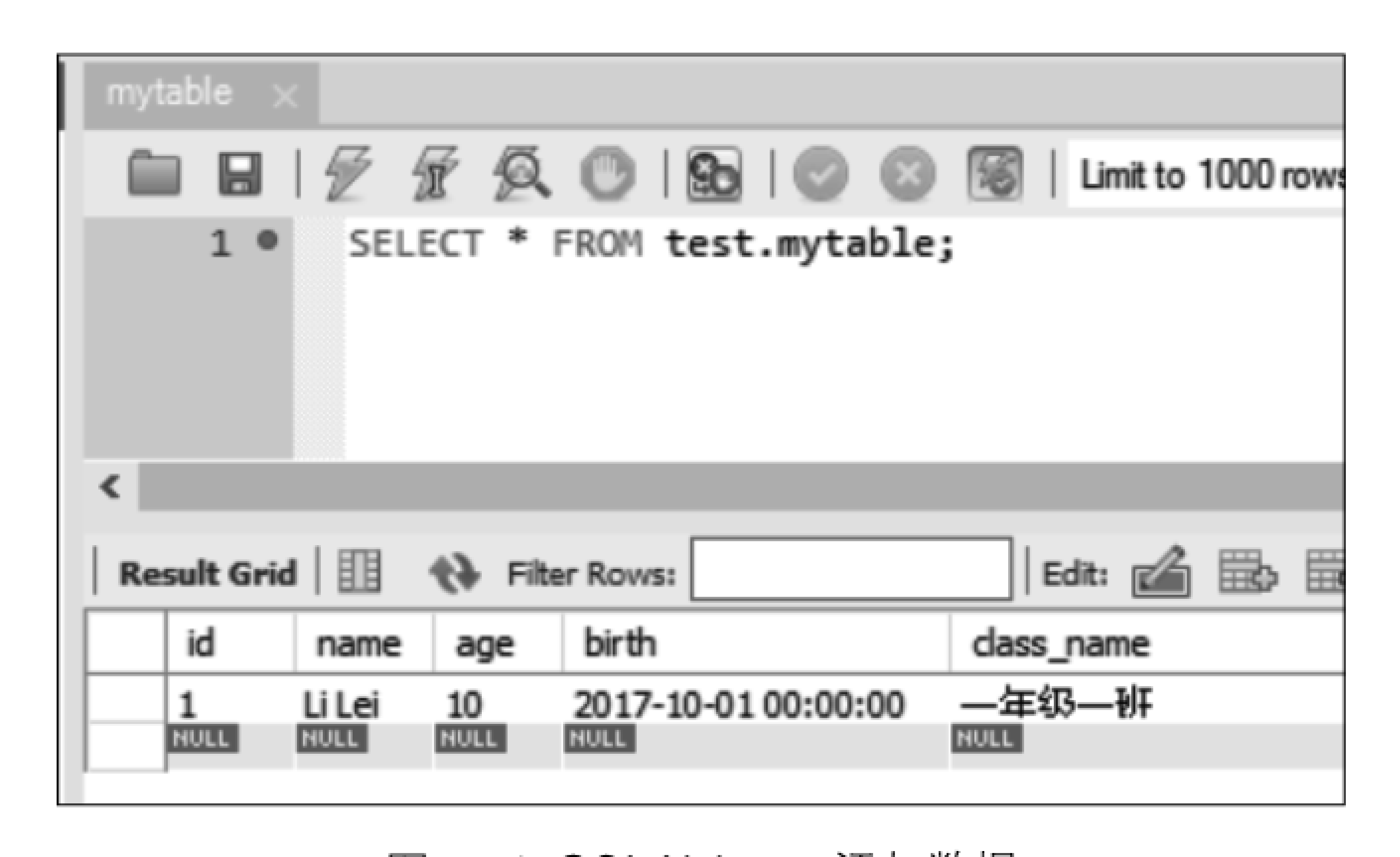


图 10-3 SQLAlchemy 添加数据

10.6 更新数据

目前，数据库中已经添加了一条数据，如果要对这条数据进行更新，SQLAlchemy 提供了以下两种更新数据的方法。

（1）使用 update 方法更新数据，代码如下：

```
session.query(mytable).filter_by(id=1).update({ mytable.age : 12})
session.commit()
session.close()
```

首先查询 mytable 表 id 为 1 的数据；然后使用 update 对这条数据进行更新，update 数据的格式是字典类型，通过键值的方式对数据进行更新；接着使用 session.coomit() 执行更新语句；最后使用 session.close() 关闭当前会话，释放资源。

如果批量更新，就可以将 filter_by(id=1) 去掉，这样能将 mytable 中 age 字段的值全部更新为 12。filter_by 相当于 SQL 语句里面的 where 条件判断。

使用赋值方式更新数据，代码如下：

```
get_data = session.query(mytable).filter_by(id=1).first()
get_data.class_name = '三年级三班'
session.commit()
session.close()
```

使用赋值方式也是将数据查询出来，生成查询对象，然后对该对象的某个属性重新赋值，最后提交到数据库执行。这种方法对批量更新不太友好，常用于单条数据的更新，若要用这种方法实现批量更新，则只能循环每条数据进行赋值更改。但这种方法对性能影响较大，批量更新使用 update() 比较合理。

运行结果如图 10-4 所示。

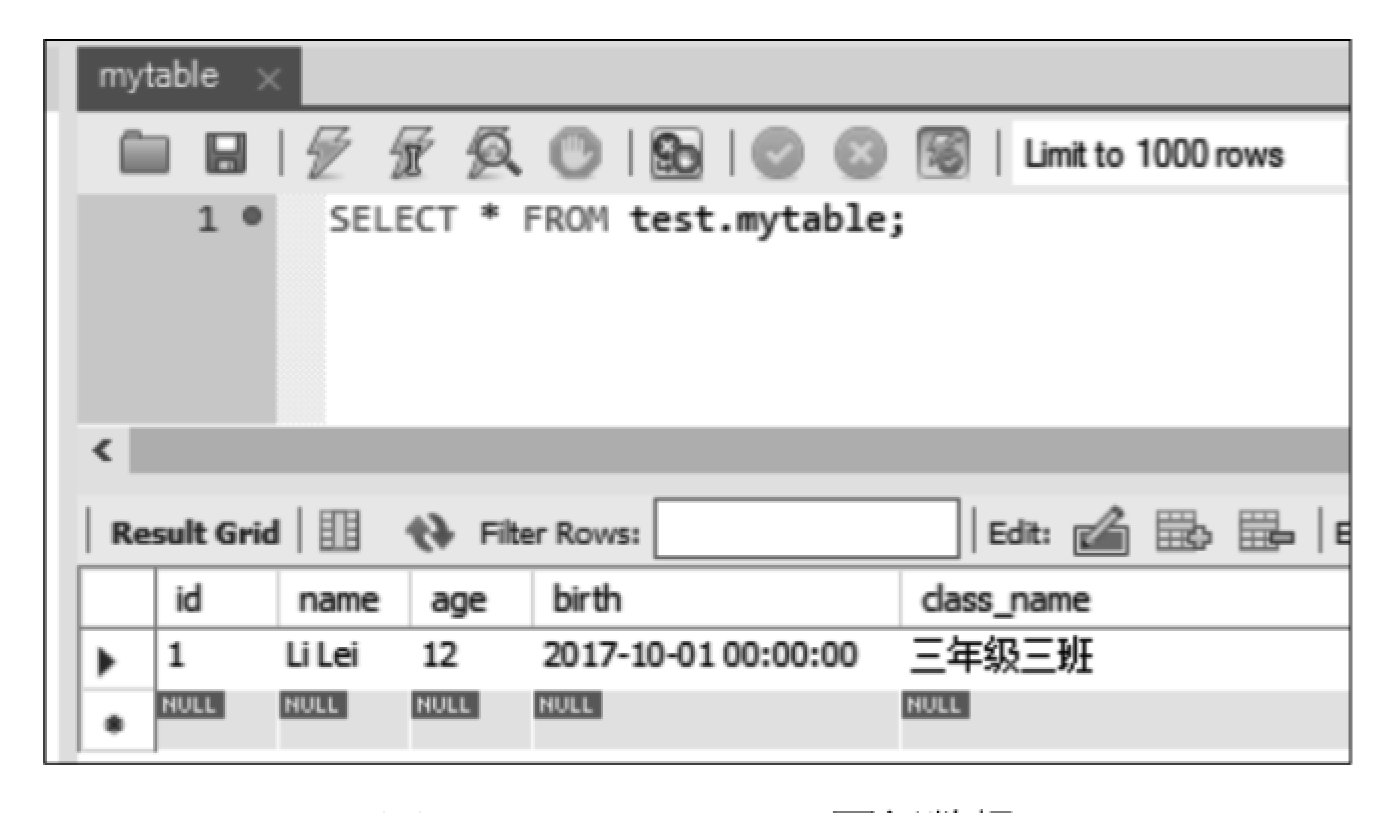
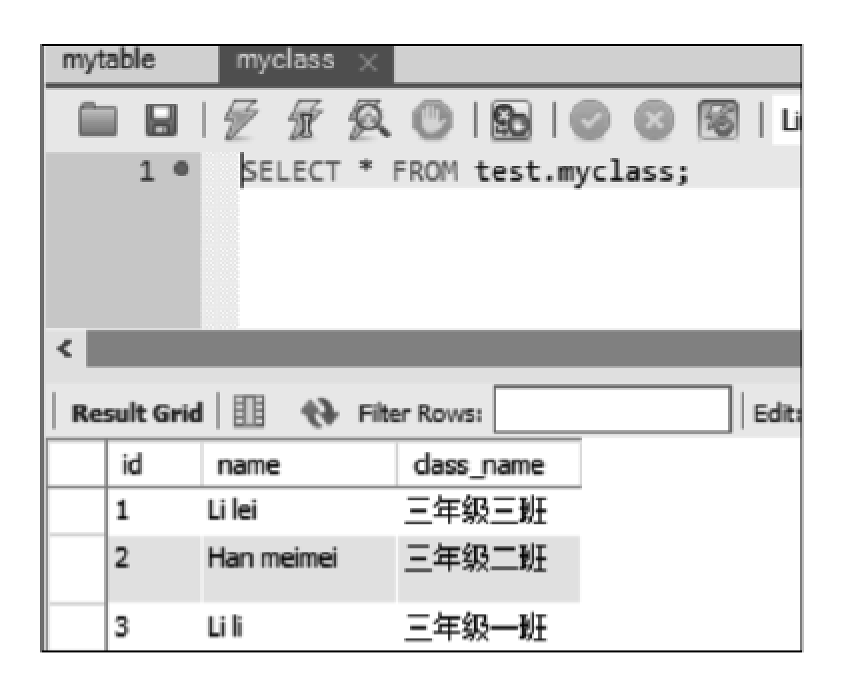


图 10-4　SQLAlchemy 更新数据

10.7 查询数据

SQLAlchemy 对数据库多种查询方式有很好的语法支持。首先对 mytable 和 myclass 加入部分数据，以便更好地讲解，如图 10-5 所示。



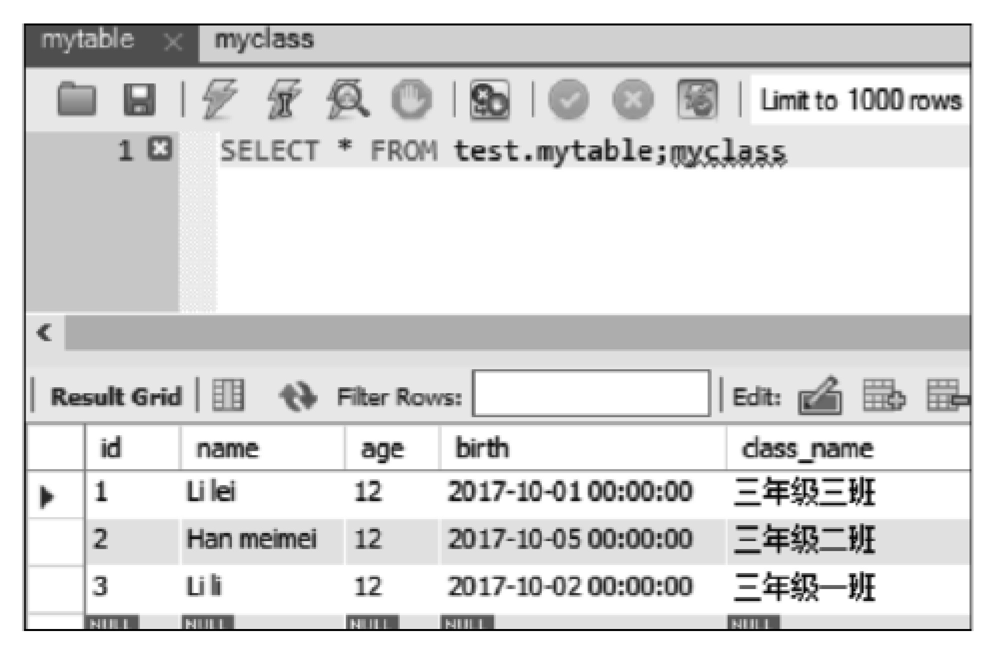


图 10-5 数据表数据内容

由图 10-5 可以看到，两个数据表已添加部分数据，查询某个数据表中数据的代码如下：

```
# 查询 myclass 全部数据
get_data = session.query(myclass).all()
for i in get_data:
    print('我的名字是：' + i.name)
    print('我的班级是：' + i.class_name)
session.close()
```

代码 session.query(myclass) 相当于 SQL 语句里面的 select * from myclass；而 all() 是将数据以列表的形式返回。

如果要查询某一字段，如 SQL 语句 select name,class_name from myclass，代码如下：

```
get_data = session.query(myclass.name, myclass.class_name).all()
for i in get_data:
    print('我的名字是：' + i.name)
    print('我的班级是：' + i.class_name)
session.close()
```

设置筛选条件，SQLAlchemy 有两种筛选方法，代码如下：

```
# 根据条件查询某条数据
# 筛选方法一：
# get_data = session.query(myclass).filter(myclass.id==1).all()
# 筛选方法二：
get_data = session.query(myclass).filter_by(id=1).all()
print('数据类型是：' + str(type(get_data)))
for i in get_data:
    print('我的名字是：' + i.name)
    print('我的班级是：' + i.class_name)
```

代码分别有两个 get_data 对象，两者的区别在于 filter 和 filter_by。

（1）字段写法：filter 筛选的字段是带类名（表名）的，而 filter_by 只需筛选字段即可。

（2）判断条件：filter 比 filter_by 多出一个等号。

（3）作用范围：filter 可以用于单表或者多表查询，而 filter_by 只能用于单表查询。

all() 方法是将查询数据以列表的形式返回，但只查询一条数据的时候，可以用 first() 返回第一条数据。代码如下：

```
get_data = session.query(myclass).filter_by(id=1).first()
print('数据类型是：' + str(type(get_data)))
print('我的名字是：' + get_data.name)
print('我的班级是：' + get_data.class_name)
```

实现多条件筛选，如 SQL 的 select * from mytable where id>1 and class_name='三年级二班'，实现方法如下：

```
get_data = session.query(mytable).filter(mytable.id >= 2,
                mytable.class_name == '三年级二班').first()
print('数据类型是：' + str(type(get_data)))
print('我的名字是：' + get_data.name)
print('我的班级是：' + get_data.class_name)
```

多条件查询只需要在查询条件中添加多个查询内容即可，每个查询内容以英文逗号隔开。如果将 SQL 语句的多条件查询“and”改成“or”，SQLAlchemy 代码如下：

```
from sqlalchemy import or_
session.query(mytable).filter(or_(mytable.id >= 2, mytable.class_
name == '三年级二班')).all()
```

如果涉及多表查询的内连接查询和外连接查询，实现代码如下：

```
# 内连接
get_data = session.query(mytable).join(myclass).filter(mytable.
class_name == '三年级二班').all()
print('数据类型是：' + str(type(get_data)))
for i in get_data:
        print('我的名字是：' + i.name)
        print('我的班级是：' + i.class_name)
# 外连接
get_data = session.query(mytable).outerjoin(
            myclass).filter(mytable.class_name=='三年级二班').all()
```

代码中的 join 和 outerjoin 与 SQL 语句中的 INNER JOIN 和 FULL OUTER JOIN 意思一致，两者之间在实现功能和性能上存在明显的差别。

一般来说，如果涉及复杂的查询语句，特别涉及多表查询和复杂的查询条件时，SQLAlchemy 还可以直接执行 SQL 语句，代码如下：

```
sql = 'select * from mytable '
session.execute(sql)
# 如果涉及更新、添加数据，就需要 session.commit()
session.commit()
```

10.8 本章小结

本章主要介绍了 ORM 框架的 SQLAlchemy 的功能和使用，SQLAlchemy 的理念是 SQL 数据库的量级和性能重要于对象集合，而对象集合的抽象又重要于表和行。

SQLAlchemy 操作数据库的流程如下。

- 连接数据库：使用 create_engine() 实现连接，需了解 create_engine() 各个参数的作用。

- 创建数据表：定义实体类映射数据表结构，通过操作类属性从而操作数据表字段。
- 创建持久化对象：引入 sessionmaker 模块，绑定已连接数据库的 engine 对象，生成会话对象 session。
- 添加数据：对实体类的属性赋值，通过 session.add() 方法添加数据，通过 session.commit() 提交到数据库。
- 使用更新数据：先查询需要修改的数据对象再更新。更新方法有修改对象属性值和使用 update() 方法更新数据。
- 查询数据：掌握 SQLAlchemy 查询语句，区分 filter_by 和 filter 的差异，理解多条件查询和多表查询。
- 执行 SQL 语句：SQLAlchemy 使用 execute() 方法执行 SQL 语句。

第 11 章 MongoDB 数据库操作

11.1 MongoDB 介绍

MongoDB 是一种基于分布式文件存储的数据库，由 C++ 语言编写，旨在为 Web 应用提供可扩展的高性能数据存储解决方案。MongoDB 是介于关系数据库和非关系数据库之间的产品，是非关系数据库中功能最丰富、最像关系数据库的数据库。MongoDB 支持的数据结构非常松散，类似于 JSON 的 BSON 格式，因此可以存储比较复杂的数据类型。MongoDB 最大的特点是支持的查询语言非常强大，其语法有点类似面向对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。

MongoDB 的特点是高性能、易部署、易使用，存储数据非常方便。主要功能特性有：

（1）面向集合存储、易存储对象类型的数据。

（2）模式自由。

（3）支持动态查询。

（4）支持完全索引，包含内部对象。

（5）支持查询。

（6）支持复制和故障恢复。

（7）使用高效的二进制数据存储，包括大型对象（如视频等）。

（8）自动处理碎片，以支持云计算层次的扩展性。

（9）支持 Ruby、Python、Java、C++、PHP、C# 等多种语言。

（10）文件存储格式为 BSON（一种 JSON 的扩展）。

（11）可通过网络访问。

所谓“面向集合”（Collection-Oriented），意思是数据被分组存储在数据集中，被称为一个集合（Collection）。每个集合在数据库中都有一个唯一的标识名，并且可以包含无限数目的文档。集合的概念类似关系型数据库（RDBMS）里的表（Table），不同的是 MongoDB 不需要定义任何模式（Schema），具有闪存高速缓存算法，能够快速识别数据库内大数据集中的热数据，提供一致的性能改进。

模式自由（Schema-Free），意味着对于存储在 MongoDB 数据库中的文件，不需要知道它的任何结构定义。如果需要，完全可以把不同结构的文件存储在同一个数据库里。

存储在集合中的文档被存储为键 - 值对的形式。键用于唯一标识一个文档，为字符串类型，而值则可以是各种复杂的文件类型。我们称这种存储形式为 BSON（Binary Serialized Document Format）。

MongoDB 已经在多个站点部署，其主要场景如下：

（1）网站实时数据处理。非常适合实时地添加、更新与查询，并具备网站实时数据存储所需的复制及高度伸缩性。

（2）缓存。由于性能很高，因此适合作为信息基础设施的缓存层。在系统重启之后，由它搭建的持久化缓存层可以避免下层的数据源过载。

（3）高伸缩性的场景。非常适合由数十或数百台服务器组成的数据库，它的路线图中已经包含对 MapReduce 引擎的内置支持。

11.2 安装及使用

使用 Python 操作 MongoDB 需要搭建开发环境，本节主要介绍在 Windows 下搭建 Python+MongoDB 环境配置。配置环境需安装 MongoDB、MongoDB 可视化工具和 Python 操作 MongoDB 模块。

11.2.1 MongoDB

MongoDB 的安装包可在官方网站下载社区版（www.mongodb.com/download-center#community），如图 11-1 所示。

下载完成之后，直接打开安装包，单击“Next”按钮按提示完成安装即可。完成安装后，进入 MongoDB 默认安装目录（C:\Program Files\MongoDB\Server\3.4），在当前目录下新建文件夹 data 和 log，分别用于存放数据库文件和 log 日志文件，再创建一个 mongo.conf 配置文件，如图 11-2 所示。

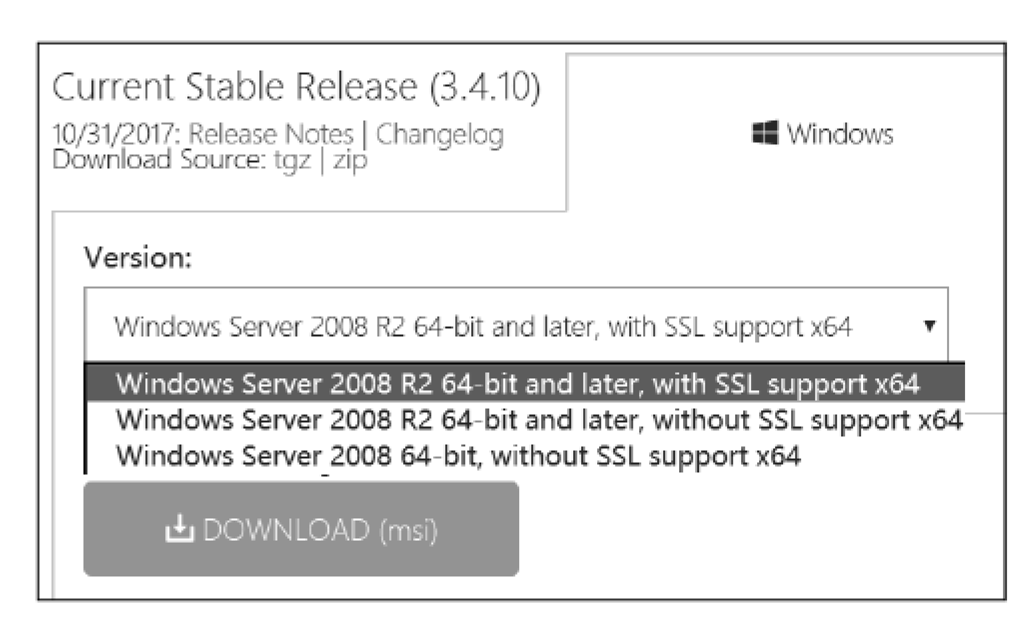


图 11-1 MongoDB 官方下载版本

Windows (C:) › Program Files › MongoDB › Server › 3.4
bin
data
log
GNU-AGPL-3.0
mongo.conf
MPL-2
README
THIRD-PARTY-NOTICES

图 11-2 MongoDB 安装目录

打开新创建的 mongo.conf，输入以下代码：

```
# 数据库文件路径
dbpath = C:\Program Files\MongoDB\Server\3.4\data
# 日志输出文件路径
logpath = C:\Program Files\MongoDB\Server\3.4\log\mongo.log
# 错误日志采用追加模式
```

```
logappend = true
# 启用日志文件，默认启用
journal = true
# 这个选项可以过滤掉一些无用的日志信息，若需要调试使用，则设置为 false
quiet = true
# 端口号，默认为 27017
port = 27017
```

代码中的数据库文件路径为新建的 data 文件夹路径（data 文件夹的路径没有硬性规定，一般默认为 MongoDB 的安装目录），日志文件路径为新建的 log 文件夹路径。写入配置信息后保存关闭文件，然后打开 CMD 窗口（终端），路径切换到图 11-2 中的 bin 目录，依次输入以下命令：

```
mongod --config "配置文件mongo.conf绝对路径" --install --serviceName "MongoDB"
net start MongoDB
```

以上命令代表将 MongoDB 数据库服务器添加到 Windows 服务，这样可免去每次手动开启 MongoDB。运行结果如图 11-3 所示。

```
C:\Program Files\MongoDB\Server\3.4\bin>mongod --config "C:\Program Files\MongoDB\Server\3.4\mongo.conf" --install --serviceName "MongoDB"

C:\Program Files\MongoDB\Server\3.4\bin>net start MongoDB
MongoDB 服务正在启动 .
MongoDB 服务已经启动成功。


C:\Program Files\MongoDB\Server\3.4\bin>
```

图 11-3 MongoDB 配置信息

完成配置设置后，在浏览器中输入 http://127.0.0.1:27017/ 验证配置是否成功，若出现如图 11-4 所示的内容，则说明配置成功。

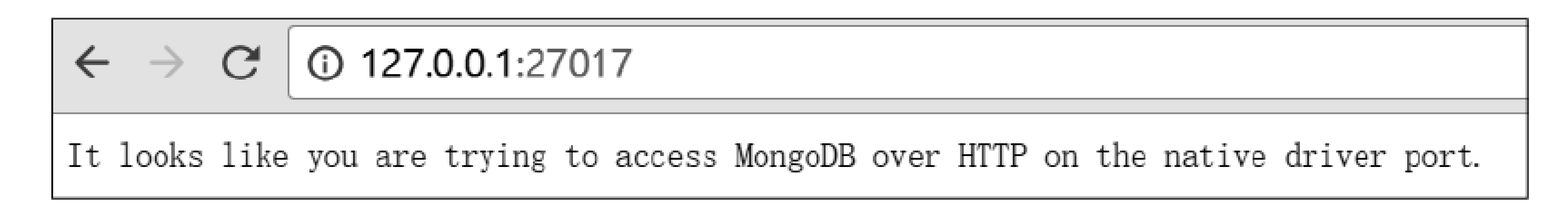


图 11-4 MongoDB 配置成功

11.2.2 MongoDB 可视化工具

可视化工具可帮助使用者快速查看数据库的使用情况，MongoDB 常用的可视化工具有 RoboMongo 和 MongoBooster。

以 RoboMongo 使用为例，官方网站下载地址为 https://robomongo.org/download，下载后运行 .exe 文件，按提示可完成安装。然后运行软件，单击“MongoDB Connections”界面中的“Create”按钮，弹出“Connections Settings”，输入 Name 和 Address 的信息：Name 为对该连接的命名，可自定义命名；Address 处分别输入数据库 IP 地址和端口。此处以本地数据库为例，如图 11-5 所示。

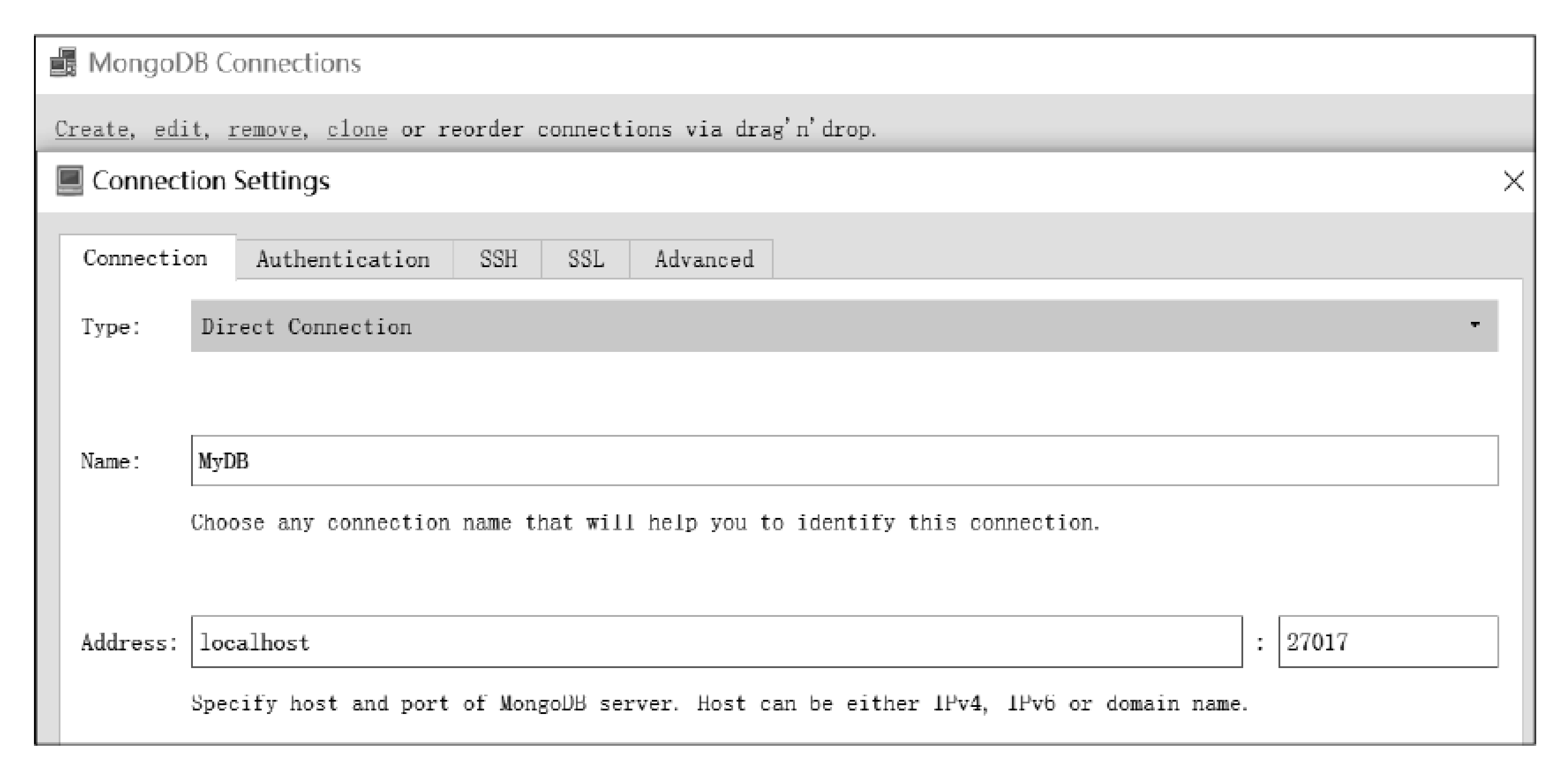


图 11-5 RoboMongo 创建数据库连接

连接数据库后，会看到数据库有一个“system”文件夹，文件夹里有“admin”和“local”数据库，两者皆属于系统数据库，如图 11-6 所示。

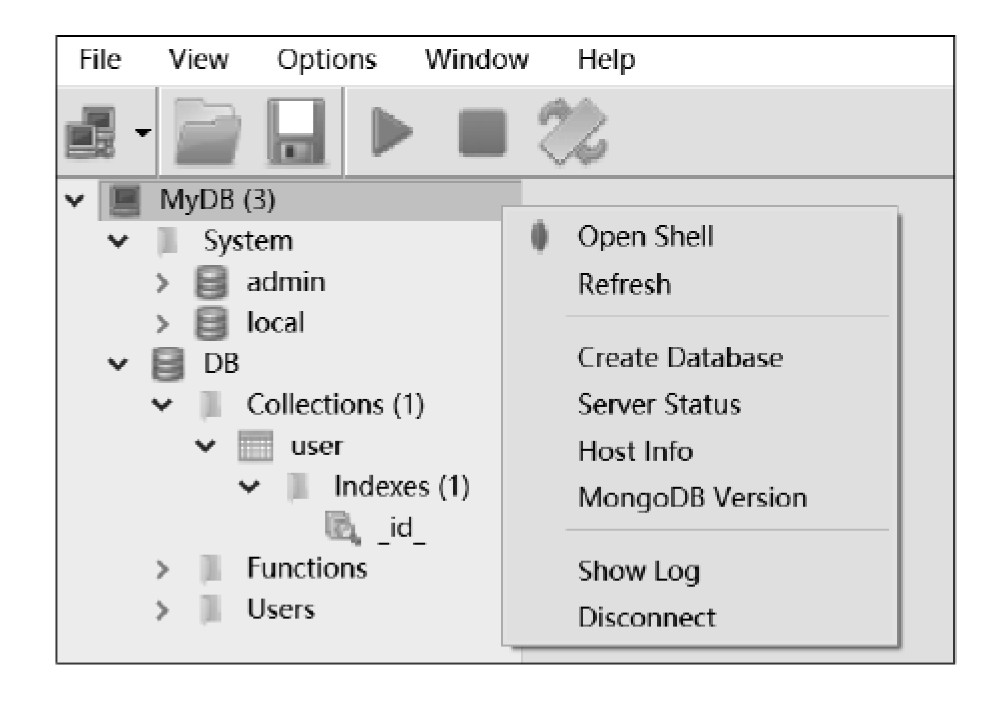


图 11-6 MongoDB 数据结构

结合图 11-6 创建数据库，方法如下：

步骤 01 右击“MyDB”，单击“Create Database”，将数据库命名为“DB”。

步骤 02 打开数据库“DB”，右击“Collections”，选择“Create Collection”，命名为“user”。新建的 user 称为集合，相当于关系数据库里面的数据表。

步骤 03 右击“user”，选择“Insert Document”。Document 代表文档内容，相当于 MySQL 里数据表中的数据。Document 是 BSON 格式，类似 JSON，如图 11-7 所示。

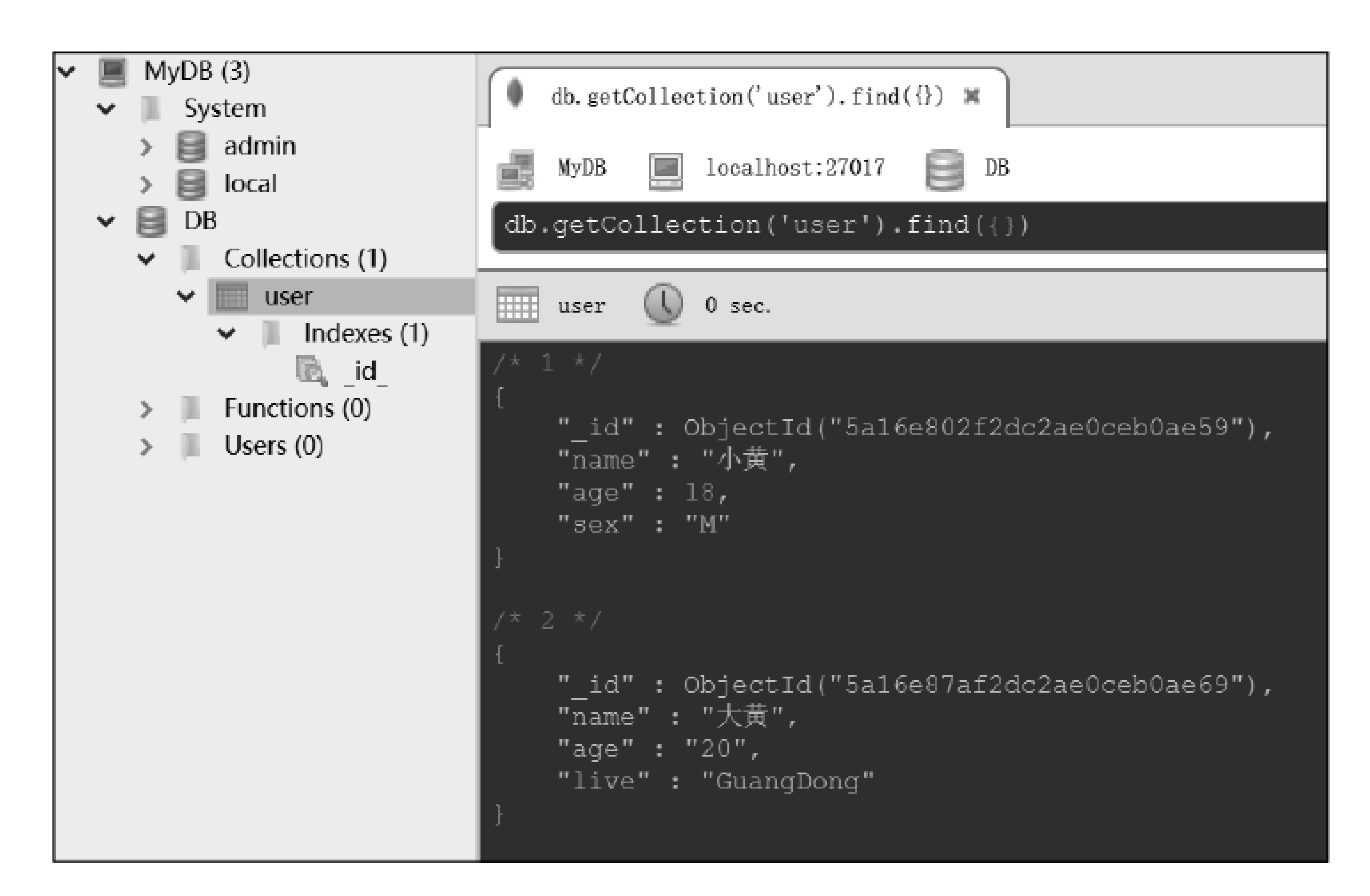


图 11-7　MongoDB 添加文档

步骤 04　集合 user 里有文件夹“Indexes”，用于实现集合的索引功能；文件夹“Functions”用于实现脚本功能；在“Users”中设定用户账号密码，用于设置访问权限。

11.2.3 PyMongo

PyMongo 是 Python 操作 MongoDB 的第三方库，有庞大的社区，功能较为稳定和完善。建议使用 pip 安装 PyMongo：

```
pip install pymongo
```

完成安装后，打开 CMD 窗口，通过导入模块测试是否安装成功：

```
>>> import pymongo
>>> pymongo.__version__
'3.5.1'
```

11.3 连接数据库

前面通过可视化工具对 MongoDB 进行了讲解，相信大家对 MongoDB 的数据结构有了一定的了解。使用 Python 实现对 MongoDB 操作的原理与连接关系式数据库一样：连接数据库→访问数据表（集合）→增删改查。

Python 连接 MongoDB 主要由 PyMongo 实现，连接代码如下：

```
import pymongo
# 创建对象，连接本地数据库
# 方法一：
client = pymongo.MongoClient()
# 方法二：
client = pymongo.MongoClient('localhost', 27017)
# 方法三：
client = MongoClient('mongodb://localhost:27017/')
# 连接 DB 数据库
db = client['DB']
# 连接集合 user，集合类似关系数据库的数据表
# 如果集合不存在，就会新建集合 user
user_collection = db.user
# 设置文档格式（文档即我们常说的数据）
```

代码使用三种方法创建数据库（client）对象，localhost 是数据库 IP 地址，27017 是数据库端口，db = client['DB'] 指向需要连接的数据库，user_collection = db.user 指向 user 集合（相当于关系数据库的数据表）。

如果数据库设置了用户验证，在连接命令上要添加验证信息：

```
import pymongo
# 用户验证方法一
client = pymongo.MongoClient()
db_auth = client.admin
db_auth.authenticate(username, password)
# 连接 DB 数据库
db = client['DB']
# 用户验证方法二
client = MongoClient('mongodb://username:password@localhost:27017/')
# 连接 DB 数据库
db = client['DB']
```

上述代码提供两种验证方式，用户验证实质上是在连接数据库的时候，将数据库用户的账号、密码添加到连接语句上实现验证登录。

11.4 添加文档

在 MongoDB 中，常用的操作有添加文档、更新文档、删除文档和查询文档。文档的数据结构和 JSON 基本一样。所有存储在集合中的数据都是 BSON 格式。BSON 是一种类似 JSON 的二进制形式的存储格式，简称 Binary JSON。

文档添加方式分别有单条添加和批量添加，实现代码如下：

```
import pymongo
import datetime
import re
# 创建对象
client = pymongo.MongoClient()
# 连接 DB 数据库
db = client['DB']
# 连接集合 user，集合类似关系数据库的数据表
# 如果集合不存在，就会新建集合 user
user_collection = db.user
# 设置文档格式（文档即我们常说的数据）
user_info = {
    "_id": 100,
    "author": " 小黄 ",
    "text": "Python 爬虫开发 ",
    "tags": ["mongodb", "python", "pymongo"],
    "date": datetime.datetime.utcnow()}

# 使用 insert_one 单条添加文档，inserted_id 获取写入后的 id
# 添加文档时，如果文档尚未包含 "_id" 键，就会自动添加 "_id"。 "_id" 的值在集合中必须是唯一的
# inserted_id 用于获取添加后的 id，若不需要，则可以去掉
user_id = user_collection.insert_one(user_info).inserted_id
print ("user id is ", user_id)

# 批量添加
user_infos = [{
    "_id": 101,
    "author": " 小黄 ",
```

```
            "text": "Python 爬虫开发 ",
            "tags": ["mongodb", "python", "pymongo"],
            "date": datetime.datetime.utcnow()},
    {
        "_id": 102,
        "author": " 小黄 _A",
            "text": "Python 爬虫开发 _A",
            "tags": {"db":"Mongodb","lan":"Python","modle":"Pymongo"},
            "date": datetime.datetime.utcnow()},
    ]
   # inserted_ids 用于获取添加后的 id，若不需要，则可以直接去掉
   user_id = user_collection.insert_many(user_infos).inserted_ids
   print ("user id is ", user_id)
```

代码实现了单条添加和批量添加，单条添加的数据是 user_info，该数据是一个字典数据结构；批量添加的数据是 uesr_infos，该数据是一个字典数据组成的列表。执行数据添加分别由 insert_one 和 insert_many 方法实现。数据添加完成后，使用 inserted_id 和 inserted_ids 可返回添加后所自动生成的 id 内容。

11.5 更新文档

更新文档同样分为单条更新和批量更新，分别由 update() 和 update_many() 实现。文档更新需要加入操作符。操作符的作用：通常文档只会有一部分要更新，利用原子的更新修改器可以使得这部分更新极为高效。MongoDB 提供了许多原子操作，比如文档的保存、修改、删除等。所谓原子操作，就是要么将这个文档保存到 MongoDB，要么没有保存到 MongoDB，不会出现查询到的文档没有保存完整的情况。更新修改器是一种特殊的键，用来指定复杂的更新操作，比如调整、增加或者删除键，还可能用于操作数组或者内嵌文档。

下面介绍常用的更新操作符。

- $set：用来指定一个键的值。如果这个键不存在，就创建它；如果存在，就执行更新。
- $unset：从文档中移除指定的键。

- $inc：修改器用来增加已有键的值，或者在键不存在时创建一个键。$inc 就是专门来增加（和减少）数字的，只能用于整数、长整数或双精度浮点数。要是用在其他类型的数据上，就会导致操作失败。
- $rename：操作符可以重命名字段名称，新的字段名称不能和文档中现有的字段名相同。如果文档中存在 A、B 字段，将 B 字段重命名为 A，$rename 会将 A 字段和值移除掉，然后将 B 字段名改为 A。
- $push：如果指定的键已经存在，就会向已有的数组末尾加入一个元素；如果指定的键不存在，就会创建一个新的数组。

如何使用操作符实现更新文档呢？例如更新上述已添加的文档的代码如下：

```
# 更新单条文档
# update(筛选条件，更新内容)。筛选条件为空，默认更新第一条文档
user_collection.update(
{},
{"$set":{"author":"小黄","text":"Python 爬虫"}}
)
# 批量更新文档，只要将方法 update 改为 update_many 即可
```

在代码中，user_collection 是 11.4 节的集合 user 对象，方法 update 有两个参数，皆为字典格式：第一个字典为筛选条件，若为空，则默认更新第一条文档；第二个字典以操作符为字典的键，更新的内容以字典格式作为字典的值。

11.6 查询文档

查询文档是使用 find() 方法产生一个查询来从 MongoDB 的集合中查询到数据。该方法与其他方法的使用大致相同，使用方法如下：

```
# 查询文档，find({"_id":101})，其中{"_id":101}为查询条件，若查询条件为空，则默认查询全部
find_value = user_collection.find({"_id":101})
print(list(find_value))
```

如果要实现多条件查询，就需要使用查询操作符：$and 和 $or，使用方法如下：

```
# AND 条件查询
find_value = user_collection.find({
"$and":[{"_id":101},{"author":" 小黄 "}]
})
print(list(find_value))
# OR 条件查询
find_value = user_collection.find({
"$or":[{"author":" 小黄 _A"},{"author":" 小黄 "}]
})
print(list(find_value))
```

方法 find() 传递字典作为查询条件，操作符 $and 和 $or 作为字典的键，字典的值是列表格式的，列表中的元素以字典形式表示，一个元素代表一个查询条件。

如果要实现大于、小于或者不等于这类比较查询，就需要使用比较查询操作符：$lt（小于）、$lte（小于或等于）、$gt（大于）、$gte（大于或等于）、$in（in，符合范围内）、$nin（not in，范围之外），使用方法如下：

```
# 如查找 id>100 而 <102，即 _id=101 的文档
find_value = user_collection.find({
"_id":{"$gt":100,"$lt":102}
})
print(list(find_value))
# 查找 id 在 [100,101]
find_value = user_collection.find({
"_id":{"$in":[100,101]}
})
print(list(find_value))
```

比较查询和多条件查询存在明显的差别：

（1）多条件查询以操作符为字典的键，比较查询以字段为字典的键。

（2）多条件查询的值是列表格式的，比较查询的值是字典格式的。

如果使用两者组成一个查询，代码如下：

```
find_value = user_collection.find({
"$and": [{"_id": {"$gt":100,"$lt":102}, {"_id": {"$in": [100,101]}}]
```

```
})
print(list(find_value))
```

从代码中可以看到，多条件查询操作符 $and 作为最外层字典的键，比较查询操作符位于最里层字典。$and 是将每个条件连接起来，主要作用于每个查询条件之间；比较查询操作符（$gt 和 $in）使条件按照某个规则成立条件判断，主要作用于每个查询条件里面。

当查询条件不明确某个值的时候，可以使用模糊匹配进行查询。在 MongoDB 中实现模糊匹配需要引用正则表达式，代码如下：

```
# 模糊查询实际上是加入正则表达式实现
# 方法一：
find_value = user_collection.find({
"author": {"$regex": ".* 小 .*"}
})
print(list(find_value))

# 方法二：
regex = re.compile(".* 小 .*")
find_value = user_collection.find({
"author": regex
})
print(list(find_value))
```

实现模糊匹配有两种不同的方式，两者都需要引用正则表达式来完成模糊功能。

方法一：使用操作符 $regex 作为字典的键，告诉数据库这个查询语句要查找字段 author 中含有“小”的内容。

方法二：re.compile 定义了一个 Pattern 实例，这是正则表达式对象，将其实例作为查询条件的值，同样也是告诉数据库需要查找字段 author 中含有“小”的内容。

我们知道 JSON 可以嵌套多个 JSON，MongoDB 的文档也是如此。当查询文档中某个字段嵌套多个文档时，如何将嵌套里面的文档作为查询条件实现文档查询呢？代码如下：

```
# 查询嵌入 / 嵌套文档
# 查询字段 "tags": {"db":"Mongodb", "lan":"Python", "modle":"Pymongo"}
```

```
# 查询嵌套字段，只需要查询嵌套里的某个值即可
find_value = user_collection.find({
"tags.db": "Mongodb"
})
print(list(find_value))
```

字段 tags 的值是一个字典类型的数据，也就是说，文档中 tags 字段的值嵌套了另一个文档，如果查询条件是“db”:“Mongodb”，而“db”属于字段 tags，可通过“tags.db”对其进行定位。如果“db”的值再嵌套一个字典，那么可用相同的方式进行下一步的定位，代码如下：

```
# 查询字段"tags": {"db": {"Mongodb":"NoSql","MySql":"Sql"},
         "lan":"Python", "modle":"Pymongo"}
find_value = user_collection.find({
"tags.db.Mongodb": "NoSql"
})
print(list(find_value))
```

11.7 本章小结

MongoDB 是一个基于分布式文件存储的数据库，旨在为 Web 应用提供可扩展的高性能数据存储解决方案，是介于关系数据库和非关系数据库之间的产品，是非关系数据库中功能最丰富的数据库。在当前的爬虫程序中，如何操作 MongoDB 也成为爬虫程序的重要内容。

在本章中，读者要重点掌握以下内容：

（1）熟悉 MongoDB 安装配置。

（2）理解 MongoDB 数据结构。

（3）掌握使用 RoboMongo 操作 MongoDB。

（4）添加文档分为单条添加和批量添加，分别由 insert_one() 和 insert_many() 实现。

（5）更新文档分为单条更新和批量更新，分别由 update() 和 update_many() 实现，并且掌握更新操作符的使用。

（6）查询文档由 find() 实现，掌握比较查询、多条件查询、模糊查询和嵌套查询。

第 12 章

项目实战：爬取淘宝商品信息

12.1 分析说明

本章讲述使用 Python 爬取淘宝商品信息，数据主要用于商家分析市场趋势，从而制定一系列营销方案。实现的功能如下：

（1）使用者提供关键字，利用淘宝搜索功能获取搜索后的数据。

（2）获取商品信息：标题、价格、销量、店铺名称和店铺所在区域。

（3）数据以文件方式存储。

功能实现依次体现了爬虫的开发流程：爬取规则→数据清洗→数据存储。整个爬虫开发需要以网站的分析作为支撑点。

我们使用 Chrome 浏览器分析淘宝网站，在 Chrome 中输入 https://www.taobao.com/ 后按回车键，进入淘宝首页，利用搜索功能搜索某关键字，例如以“四件套”为关键字进行搜索。打开浏览器的开发者工具，单击 Network 标签，如果没有捕捉到网站的请求信息，那么可以刷新网页重新捕捉，如图 12-1 所示。



图 12-1 使用 Chrome 获取淘宝网页信息

一个网页的请求信息包含多种请求类型，网页数据主要在 Doc、XHR 和 JS 分类标签里面找到。在不确定具体请求信息的情况下，通常先从请求的响应内容查找是否有我们需要的数据，如图 12-2 所示。

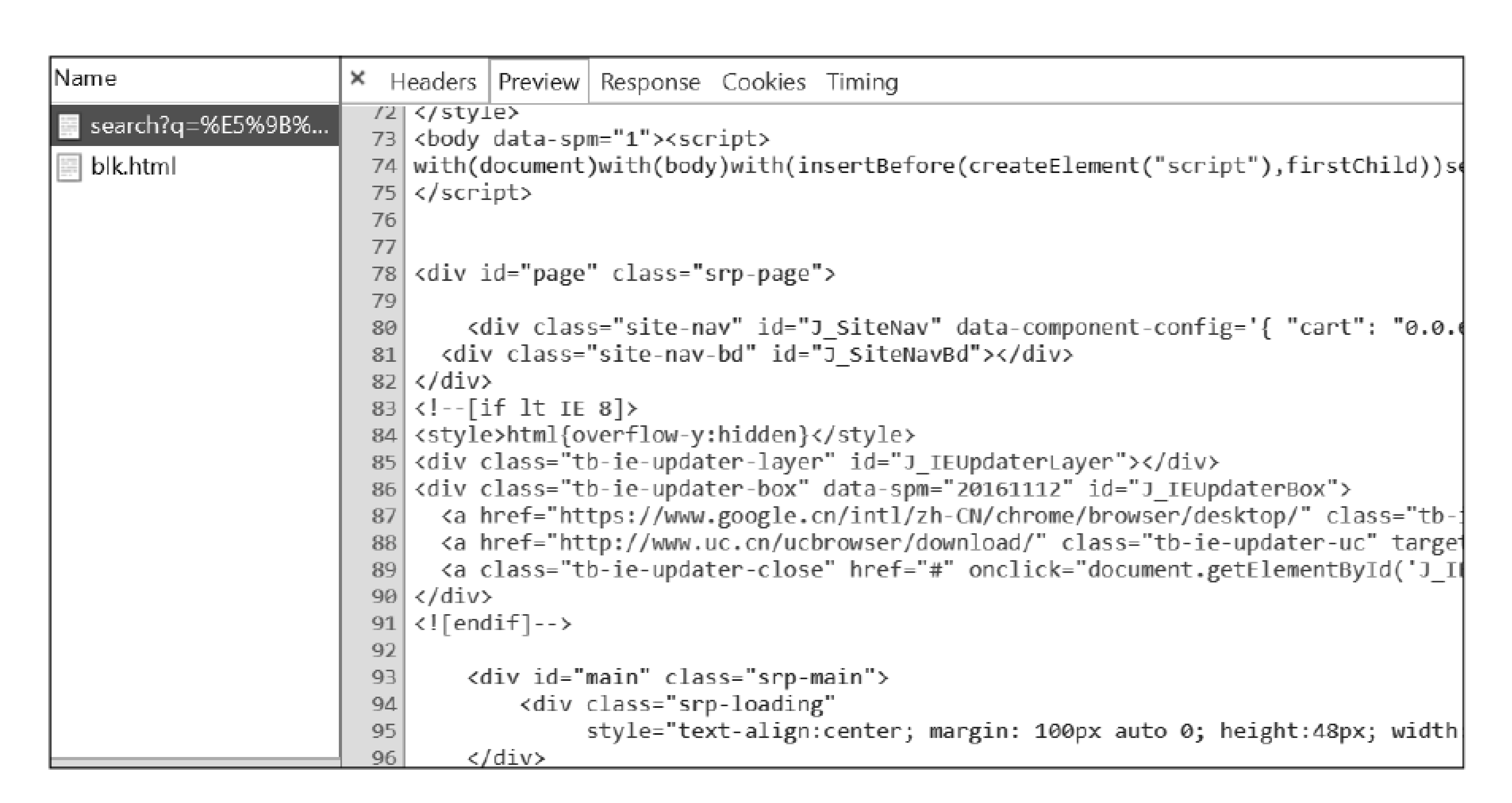


图 12-2 Doc 标签下的响应内容

从图 12-2 中得知，Doc 标签的响应内容大多数是一些 JS 脚步和简单的 HTML，并没有我们所需的数据，也就是说，浏览器访问当前的 URL 所返回 HTML 文件是不生成淘宝商品信息的。

在响应的 HTML 文件中找不到所需数据，那么数据的生成可能来源于 Ajax 请求后台，再通过前端渲染在网页上。

单击 XHR 标签，发现有一个 Ajax 请求，查看该请求的详细信息，如图 12-3 所示。

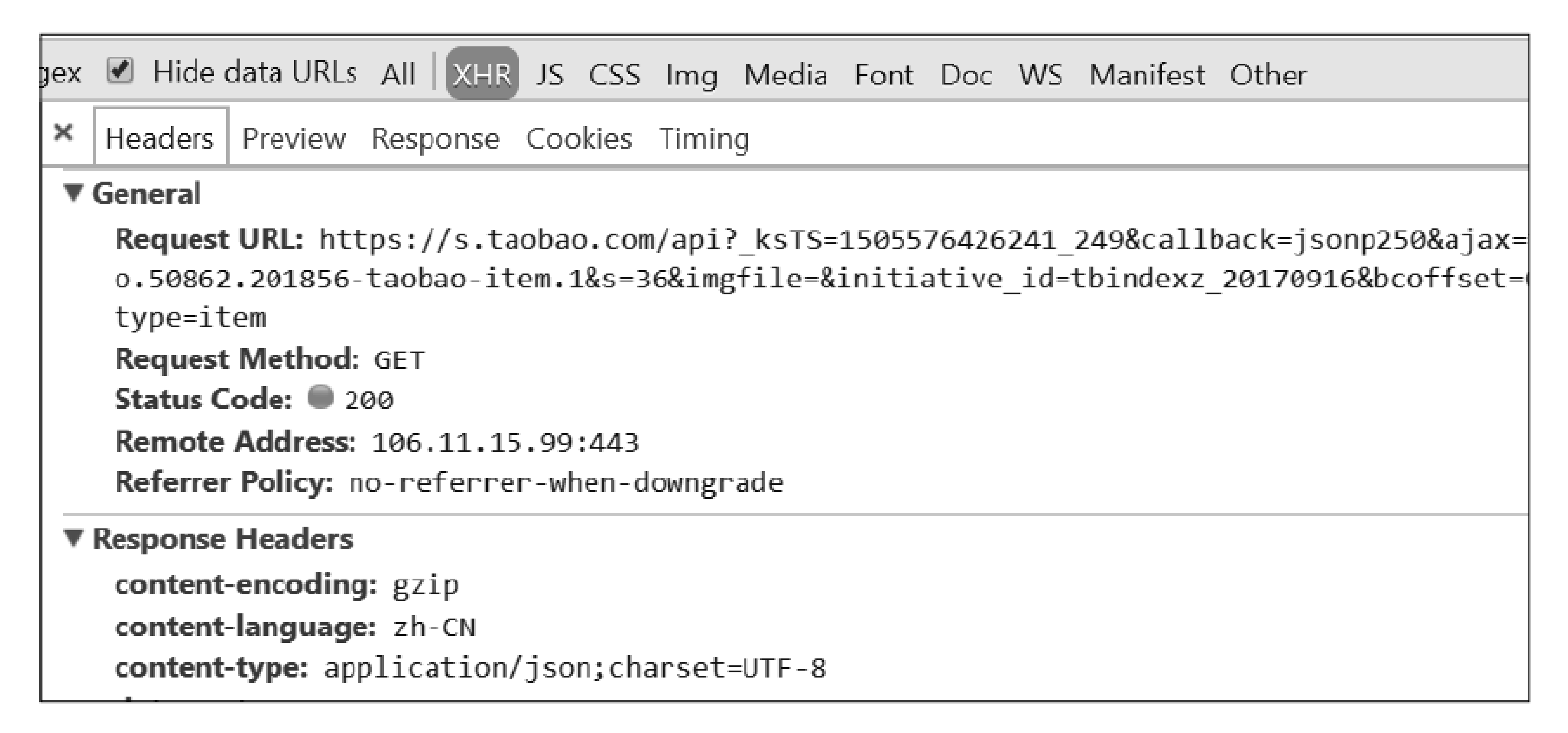


图 12-3　Ajax 请求信息

单击 Preview 查看该 URL 的响应内容，发现响应的数据是 JSON 格式的，而且数据内容是当前网页上的淘宝商品信息，商品信息的标题、价格、销量、店铺名称和店铺所在区域分别对应数据的 raw_title、view_price、view_sales、nick 和 item_loc，如图 12-4 所示。

```
ex  Hide data URLs  All  XHR  JS  CSS  Img  Media  Font  Doc  WS  Manifest  Other
×  Headers  Preview  Response  Cookies  Timing
▼jsonp250({,…});
 ▼API.CustomizedApi: {itemlist: {auctions: [{,…}, {,…}, {,…}, {,…},…], trace: "auction_customized", query: "四件套"}}
  ▼itemlist: {auctions: [{,…}, {,…}, {,…}, {,…},…], trace: "auction_customized", query: "四件套"}
   ▼auctions: [{,…}, {,…}, {,…}, {,…},…]
    ▼0: {,…}
      category: "50008779"
      comment_count: "1565"
      comment_url: "//item.taobao.com/item.htm?id=540141622303&ns=1&abbucket=4&on_comment=1"
      detail_url: "//item.taobao.com/item.htm?id=540141622303&ns=1&abbucket=4#detail"
     ▶i2iTags: {samestyle: {url: "/search?type=samestyle&app=i2i&rec_type=1&uniqpid=-1440327761&nid=540141622303"},…}
     ▶icon: [{title: "卖家提供以下贴心服务：", dom_class: "icon-service-fuwu", position: "99", show_type: "1",…},…]
      item_loc: "江苏 南通"
      nick: "爱享乐家纺"
      nid: "540141622303"
      p4pTags: []
      pic_url: "//g-search2.alicdn.com/img/bao/uploaded/i4/i2/2874154753/TB23JA_oVXXXXcuXpXXXXXXXXXX_!!2874154753.jpg"
      pid: "-1440327761"
      raw_title: "纯色四件套全棉1.8m床单被套纯棉素色简约1.5米2.0m双人床上用品"
      risk: ""
      shopLink: "//store.taobao.com/shop/view_shop.htm?user_number_id=2874154753"
     ▶shopcard: {levelClasses: [{levelClass: "icon-supple-level-zuan"}, {levelClass: "icon-supple-level-zuan"},…],…}
      title: "纯色<span class=H>四件</span><span class=H>套</span>全棉1.8m床单被<span class=H>套</span>纯棉素色简约1.5米2.0m双人床上用品"
```

图 12-4　Ajax 请求响应内容

从图 12-4 得知，请求链接信息如下：

```
https://s.taobao.com/api?_ksTS=1505577563463_249&callback=jsonp80
4&ajax=true&m=customized&stat
```

s_click=search_radio_all:1&q= 四件套 &s=36&imgfile=&initiative_id=staobaoz_20170917&bcoffset=0&js=1
&ie=utf8&rn=c78bd6f024943219080010270551 61f1

请求链接的信息很长，某些请求参数看起来杂乱无章，有可能存在一些非必要的请求参数。为了让 URL 进一步简化，我们将 URL 复制到 Chrome 上访问一次，浏览器直接返回响应内容，然后每次删除一个参数后重新访问，查看是否得到相同的响应内容。通过筛选和删减，得到最后的 URL，如图 12-5 所示。

https://s.taobao.com/api?callback=jsonp804&m=customized&q=四件套&s=0

jsonp804({"API.CustomizedApi":{"itemlist":{"auctions":[{"i2iTags":{"samestyle":{"url":"/search?typ
1121013486\u0026nid\u003d527314661809"},"similar":{"url":"/search?type\u003dsimilar\u0026app\u003d
1121013486\u0026nid\u003d527314661809"}},"p4pTags":[],"nid":"527314661809","category":"50008779","
\u003c/span\u003e\u003cspan class\u003dH\u003e套\u003c/span\u003e 艾莎诺雅 清新淡雅床上用品","raw_
search3.alicdn.com/img/bao/uploaded/i4/i2/TB19avZLFXXXXbtXFXXXXXXXXXX_!!0-item_pic.jpg","detail_ur
id\u003d527314661809\u0026ns\u003d1\u0026abbucket\u003d4","view_price":"699.00","view_fee":"6.00",
款","comment_count":"236","user_id":"92042735","nick":"水星旗舰店","shopcard":{"levelClasses":[{"1
{"levelClass":"icon-supple-level-jinguan"}],"isTmall":true,"delivery":[484,1,1404],"description":[
[484,1,398],"encryptedUserId":"UOFIWMmIuvG8T","sellerCredit":18,"totalRate":10000},"icon":[{"title
tianmao","position":"1","show_type":"0","icon_category":"baobei","outer_text":"0","html":"","icon_
贝","url":"//www.tmall.com/"},{"title":"保险理赔","dom_class":"icon-service-baoxian","position":"9
service-baoxian","trace":"srpservice","innerText":"分组-保险理赔","iconPopupComplex":{"popup_title

图 12-5 简化请求链接

从简化的 URL 看出，有两个参数可以动态设置来获取不同商品的信息。

（1）q= 四件套：这是关键字搜索，可以根据这个变量获取不同类型的商品信息。

（2）s=36：这是页数设置，如果对请求链接的页数从 0 到 36 依次访问，每页 12 条信息，那么共 432 条商品信息。再观察每页之间的数据，比如页数从 0 到 1，24 条数据中有 22 条是重复的。也就说，每页数据是去掉上一页的第一条数据，再从末端新增一条数据。

12.2 功能实现

根据对网站的分析获取单个关键字搜索的单页商品信息，代码如下：

```
import requests
import json
```

```
    url = 'https://s.taobao.com/api?callback=jsonp804&m=
           customized&q= 四件套 &s=0'
    r = requests.get(url)
    response = r.text
    # 截取成标准的 JSON 格式
    response = response.split('(')[1].split(')')[0]
    # 读取 JSON
    response_dict = json.loads(response)
    # 定位到商品信息列表
    response_auctions_info = response_dict['API.CustomizedApi']
['itemlist']['auctions']
```

由于 Ajax 返回的数据是字符串格式的，在返回的值 jsonp804(XXX) 中，XXX 部分是 JSON 数据格式，因此先用字符串 split() 截取 XXX 部分，然后将 XXX 部分由字符串转换成 JSON 格式读取。

如果要获取多页数据，可以在上述代码中加入一个循环功能，实现代码如下：

```
    for p in range(88):
        url = 'https://s.taobao.com/api?callback=jsonp804&m=
                       customized&q= 四件套 &s=%s' % (p)
        r = requests.get(url)
        response = r.text
        response = response.split('(')[1].split(')')[0]
        response_dict = json.loads(response)
        # 商品信息
        response_auctions_info = response_dict['API.CustomizedApi']
['itemlist']['auctions']
```

上述代码只能获取单个关键字搜索的商品信息，如果要实现多个关键字功能，就可以在上述代码中再多加一个循环，代码如下：

```
    for k in [' 四件套 ', ' 手机壳 ']:
        for p in range(88):
            url = 'https://s.taobao.com/api?callback=jsonp804&m=
                   customized&q=%s&s=%s' % (k, p)
            r = requests.get(url)
            response = r.text
```

```
        response = response.split('(')[1].split(')')[0]
        response_dict = json.loads(response)
        # 商品信息
        response_auctions_info = response_dict['API.CustomizedApi']
['itemlist']['auctions']
```

12.3 数据存储

本节的数据存储主要选择文本文档，原因是商品数据是不断变化的，具有时效性。如果选择存放在数据库中，可能过了几天，这些数据就没有参考价值了，所以选择存放在文件中较为合适。以存储到 CSV 文件为例，代码如下：

```
def get_auctions_info(response_auctions_info, file_name):
    with open(file_name, 'a', newline='') as csvfile:
        # 生成 CSV 对象，用于写入 CSV 文件
        writer = csv.writer(csvfile)
        for i in response_auctions_info:
            # 判断是否数据已经记录
            if str(i['raw_title']) not in auctions_distinct:
                # 写入数据
                writer.writerow([i['raw_title'], i['view_price'],
                                i['view_sales'], i['nick'],
                                i['item_loc']])
                auctions_distinct.append(str(i['raw_title']))
        csvfile.close()
```

get_auctions_info() 方法实现了将数据写入 CSV 文件，调用该方法应传入参数 response_auctions_info 和 file_name，分别是商品信息列表和 CSV 文件路径。但该文件并没有对 CSV 文件设置表头，所以在开始获取数据之前，应生成对应的 CSV 文件并设定其表头。

综合上述条件，整个项目代码如下：

```
import requests
import json
import csv
```

```
# 定义全局变量，用于判断数据是否已经记录
global auctions_distinct
auctions_distinct = []

def get_auctions_info(response_auctions_info, file_name):
    with open(file_name, 'a', newline='') as csvfile:
        # 生成 csv 对象，用于写入 CSV 文件
        writer = csv.writer(csvfile)
        for i in response_auctions_info:
            # 判断是否数据已经记录
            if str(i['raw_title']) not in auctions_distinct:
                # 写入数据
                writer.writerow([i['raw_title'], i['view_price'],
                                 i['view_sales'], i['nick'],
                                 i['item_loc']])
                auctions_distinct.append(str(i['raw_title']))
        csvfile.close()

if __name__ == '__main__':
    for k in ['四件套', '手机壳']:
        # 新建 csv 文件，每循环一个关键字会生成其对应的 CSV 文件
        file_name = k + '.csv'
        with open(file_name, 'w', newline='') as csvfile:
            writer = csv.writer(csvfile)
            # 写入表头信息
            writer.writerow(['标题', '价格', '销量', '店铺', '区域'])
            csvfile.close()
        # 循环次数可以根据实际自行设定
        for p in range(88):
            url = 'https://s.taobao.com/api?callback=jsonp804&m=
                  customized&q=%s&s=%s' % (
                k, p)
            r = requests.get(url)
            response = r.text
            response = response.split('(')[1].split(')')[0]
            response_dict = json.loads(response)
            response_auctions_info = response_dict['API.CustomizedApi']
```

```
                                        ['itemlist']['auctions']
            # 调用函数 get_auctions_info 写入商品信息
            get_auctions_info(response_auctions_info, file_name)
    print('获取数据量为：' + len(auctions_distinct))
```

12.4 本章小结

本章主要介绍抓取淘宝网站的商品信息，是一个比较简单的爬虫程序。

该项目案例主要实现的功能如下：

（1）使用者提供关键字，利用淘宝搜索功能获取搜索后的数据。

（2）获取商品信息：标题、价格、销量、店铺名称和店铺所在区域。

（3）数据以文件方式存储。

从整个项目开发的角度分析，本项目最大的特点有以下 3 点：

（1）对请求链接的简化，去除无用的请求参数。

（2）分析 URL 的请求参数含义以及响应内容的数据规律。

（3）数据存储的去重判断。本案例以商品的标题判断去重，判断条件可自行修改。

建议读者掌握本项目案例的实现方法并了解其特点，同时通过上机演练实现爬虫程序的正确运行。

第 13 章

项目实战：分布式爬虫——QQ 音乐

13.1 分析说明

现在的音乐类网站仅提供歌曲在线免费试听，如果下载歌曲，往往要收取版权费用，但通过爬虫可绕开这类收费问题，可以直接下载我们所需要的歌曲。

本章以 QQ 音乐为爬取对象，爬取范围是全站的歌曲信息，爬取方式是在歌手列表下获取每一位歌手的全部歌曲。由于爬取的数量较大，还会使用异步编程实现分布式爬虫开发，提高爬虫效率。

整个爬虫项目按功能分为爬虫规则和数据入库，分别对应文件 music.py 和 music_db.py。

爬虫规则大致如下：

在歌手列表（https://y.qq.com/portal/singer_list.html）中按照字母类别对歌手进行分类，遍历每个分类下的每位歌手页面，然后获取每位歌手页面下的全部歌曲信息。根据该设计方案列出遍历次数：

（1）遍历每个歌手的歌曲页数。

（2）遍历每个字母分类的每页歌手信息。

（3）遍历每个字母分类的歌手总页数。

（4）遍历 26 个字母分类的歌手列表。

在功能上至少需要实现 4 次遍历，但实际开发中往往比这个次数要多。统计遍历次数，主要能让开发者对项目开发有整体的设计逻辑。

项目开发使用模块化设计思想，对整个项目模块的划分如下：

（1）歌曲下载。

（2）歌手信息和歌曲信息。

（3）字母分类下的歌手列表。

（4）全站歌手列表。

13.2 歌曲下载

下载歌曲前，先要找到歌曲的相关信息，才能够确定歌曲的下载链接。以 QQ 音乐中的某一首歌曲为例进行介绍（https://y.qq.com/n/yqq/song/003OUlho2HcRHC.html），在 Chrome 浏览器的网址栏输入网址后，打开开发者工具，如图 13-1 所示。

从图 13-1 分析，服务器返回的 HTML 信息（Doc 标签）只找到歌曲的少量信息，有可能歌曲信息不是由后台返回的，而是通过 Ajax 请求生成的。为了进一步求证，单击 XHR 标签，发现该标签的请求信息只是一些广告信息，说明数据也不在 XHR 标签生成。最后单击 JS 标签，发现有较多请求信息，分别查找每个请求所返回的响应内容，在某个请求中可以找到歌曲信息，如图 13-2 所示。



图 13-1　开发者工具

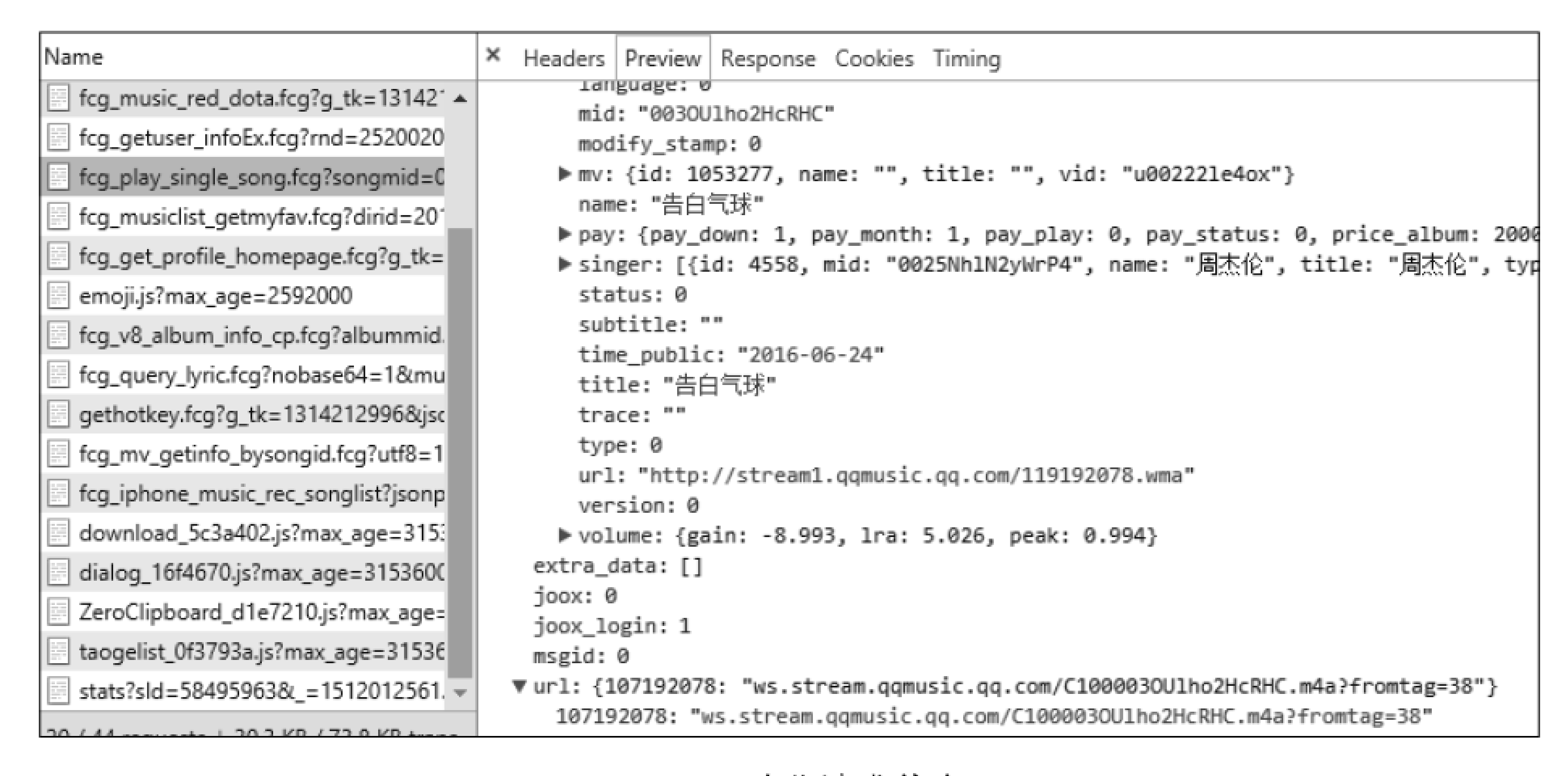


图 13-2 歌曲请求信息

从请求的响应内容中得到两个 URL 信息，分别是：

```
http://stream1.qqmusic.qq.com/119192078.wma
ws.stream.qqmusic.qq.com/C100003OUlho2HcRHC.m4a?fromtag=38
```

在浏览器中分别对两个 URL 进行访问，发现前者出现 404 报错信息，后者是一个音频文件，也就是我们所需的歌曲文件。

从后者的 URL 结构分析得知：C100003OUlho2HcRHC 用于标识歌曲的唯一性。而在该请求的响应内容中发现 mid 值是 003OUlho2HcRHC，对比发现

C100003OUlho2HcRHC 由 C100 和 003OUlho2HcRHC 组成，C100 是固定不变的，只要取到每首歌的 mid 就能确定歌曲下载链接。实现代码如下：

```
import requests
songmid = 'C100' + '003OUlho2HcRHC'
url = 'http://ws.stream.qqmusic.qq.com/%s.m4a?fromtag=38' %(songmid)
r = requests.get(url)
f = open(songmid + '.m4a', 'wb')
f.write(r.content)
f.close()
```

下载的歌曲文件的大小只有 1MB 左右，说明歌曲的音质不太好，试问品质这么差的歌曲怎么能满足广大群众的需求呢？

对此，我们不采取这种下载方式，为了寻找更优质的歌曲文件，在线试听当前歌曲，试着能否在播放页面上找到下载链接。无论在歌曲播放页面（https://y.qq.com/portal/player.html）播放什么歌曲，其 URL 都不会有变化，说明歌曲切换是由 Ajax 请求后台实现的。

在播放页中，单击开发者工具的 Media 标签，看到一个音频信息，在浏览器中访问请求链接并下载，发现下载的音频文件大小有 2.7MB，相比之前下载的歌曲文件，其音质更符合我们所需，如图 13-3 所示。

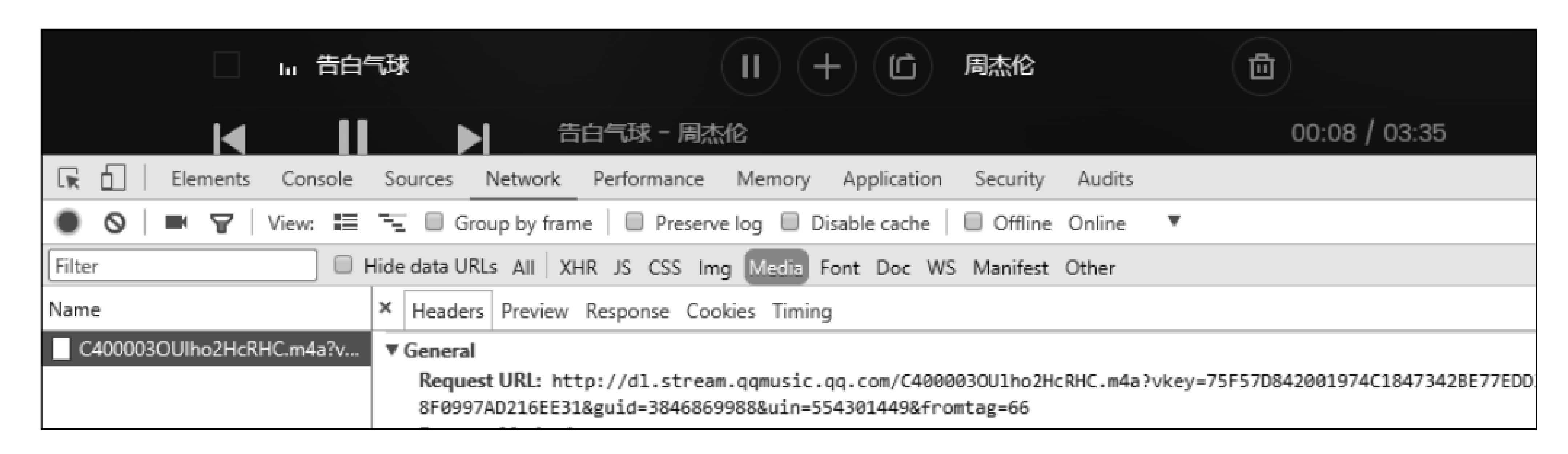


图 13-3 媒体请求信息

为方便分析 URL 构成，以代码形式展现：

```
http://dl.stream.qqmusic.qq.com/C400003OUlho2HcRHC.m4a?vkey=7E327
8292B2B5944B8B5A234FC865DFB71
54924FCE503E921E244617C8C791B55D07FFBB7D67CBFB393A88645DEB5B5BC58
835C5428D69D3&guid=3846869988&uin=554301449&fromtag=66
```

图 13-3 的请求链接分析如下：

（1）C400003OUlho2HcRHC 是由 C400+003OUlho2HcRHC 组成的，003OUlho2HcRHC 是图 13-2 的 mid 值。

（2）参数 uin 是用户的 QQ 号码，参数 fromtag 的值固定不变。

（3）参数 vkey 和 guid 无法得知由来，可能是从其他请求信息生成的，从 Doc、XHR 和 JS 依次查找参数值。最终在 JS 标签某个请求的响应内容中找出参数 vkey，如图 13-4 所示。



图 13-4　响应内容

对图 13-4 的请求链接进行分析，以代码形式展现：

```
https://c.y.qq.com/base/fcgi-bin/fcg_music_express_mobile3.fcg?
g_tk=1314212996&jsonpCallback=MusicJSONCallback04508022343182527&
loginUin=554301449&hostUin=0&format=json&inCharset=utf8&outCharset=u
tf-8&notice=0&platform=yqq&needNewCode=0&cid=205361747&
callback=MusicJSONCallback04508022343182527&uin=554301449&songmid
=003OUlho2HcRHC&filename=C400003OUlho2HcRHC.m4a&guid=3846869988
```

进一步简化图 13-4 的 URL，可将 URL 在浏览器中访问，每次删除或修改 URL 的请求参数，观察浏览器返回的响应内容是否和前一次一致，最终 URL 优化如下：

```
https://c.y.qq.com/base/fcgi-bin/fcg_music_express_mobile3.fcg?
loginUin=0&hostUin=0&cid=205361747&uin=0&songmid=003OUlho2HcRHC&f
ilename=C400003OUlho2HcRHC.m4a&guid=0
```

通过对图 13-3 和图 13-4 的 URL 和响应内容分析发现：

（1）两者的请求参数 guid 和 uin 的值必须保持一致，参数值可自行设置。

（2）图 13-3 的请求参数 vkey 来源于图 13-4 响应内容的 vkey。

（3）图 13-3 请求链接的 C400003OUlho2HcRHC.m4a 来源于图 13-4 响应内容的 filename。

综合上述分析，最终代码如下：

```
import requests
# 请求头
headers = {'User-Agent':
            'Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0)
            Gecko/20100101 Firefox/41.0'
            }
# 创建 session 会话
session = requests.session()
# 下载歌曲
def download(songmid):
    filename = 'C400' + songmid
    # 获取 vkey
    url = 'https://c.y.qq.com/base/fcgi-bin/fcg_music_express_
mobile3.fcg?loginUin=0&hostUin=0'
    &cid=205361747&uin=0&songmid=%s&filename=%s.m4a&guid=0' %
(songmid, filename)
    r = session.get(url, headers=headers)
    vkey = r.json()['data']['items'][0]['vkey']
    # 下载歌曲
    url = 'http://dl.stream.qqmusic.qq.com/%s.m4a?vkey=%s&guid=0&
uin=0&fromtag=66' %
    (filename, vkey)
    r = session.get(url, headers=headers)
    # 保存在当前目录下的 song 文件夹
    f = open('song/' + songmid + '.m4a', 'wb')
    f.write(r.content)
    f.close()

if __name__ == '__main__':
    songmid = '003OUlho2HcRHC'
download(songmid)
```

上述代码中，将下载歌曲以 download() 方法定义，参数是歌曲的 songmid，下载

的歌曲文件以歌曲的 songmid 命名。运行代码，可以看到下载的音频文件有 2.7MB，试听两者，发现后者比前者更优质。

13.3 歌手和歌曲信息

前面已实现歌曲下载功能，接着批量获取歌曲信息，实现批量下载歌曲。根据项目设计，歌曲信息是在歌手页面获取的，以周杰伦（y.qq.com/n/yqq/singer/0025NhlN2yWrP4.html#tab=song）为例，使用开发者工具分析歌手页面，分别在 Doc、XHR 和 JS 里使用“Ctrl+F”快速查找某一首歌曲信息。最终在 JS 的某一条请求中找到歌曲信息，如图 13-5 所示。

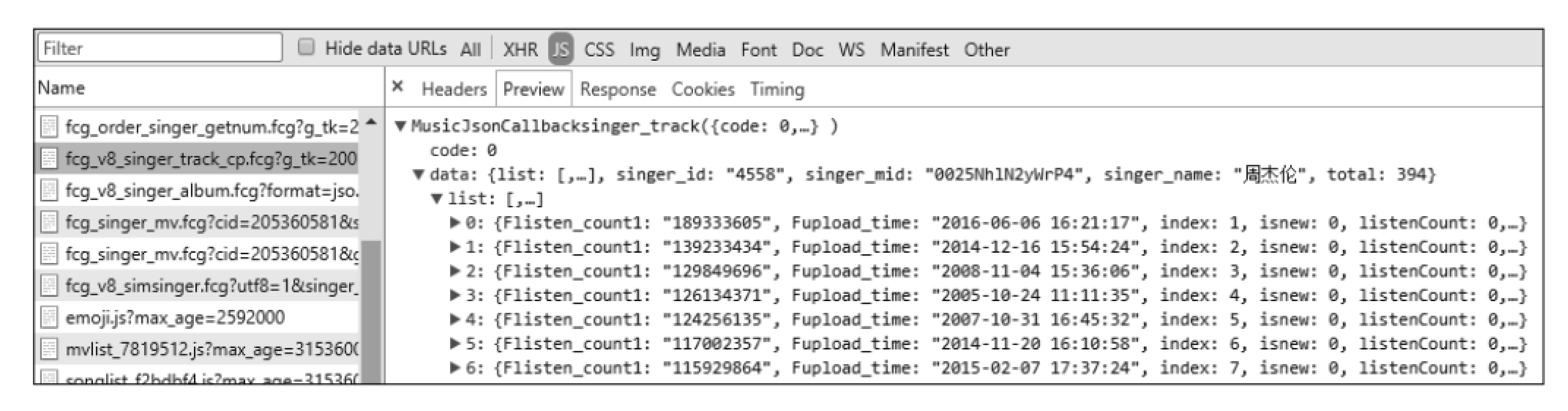


图 13-5　歌手页面的歌曲信息

从图 13-5 分析可得：

（1）total 是当前歌手的全部歌曲数目。

（2）list 是歌曲信息列表，每页共 30 首歌曲，对某首歌的信息进行分析，在信息中找到歌曲标识符、歌名、所属专辑和时长，分别对应 songmid、songname、albumname 和 interval，如图 13-6 所示。

对请求链接进行分析，以代码形式展现：

```
    https://c.y.qq.com/v8/fcg-bin/fcg_v8_singer_track_cp.fcg?g_
tk=2005119861&jsonpCallback=
    MusicJSONCallbacksinger_track&loginUin=554301449&hostUin=0&format
=jsonp&inCharset=utf8
    &outCharset=utf-8&notice=0&platform=yqq&needNewCode=0&singermid=0
025NhlN2yWrP4&order=listen
    &begin=0&num=30&songstatus=1
```

```
▼list: [,…]
  ▼0: {Flisten_count1: "189333605", Fupload_time: "2016-06-06 16:21:17", index: 1, isnew: 0, listenCount: 0,…}
     Flisten_count1: "189333605"
     Fupload_time: "2016-06-06 16:21:17"
     index: 1
     isnew: 0
     listenCount: 0
    ▼musicData: {albumdesc: "", albumid: 1458791, albummid: "003RMaRI1iFoYd", albumname: "周杰伦的床边故事", alertid: 100002,…}
       albumdesc: ""
       albumid: 1458791
       albummid: "003RMaRI1iFoYd"
       albumname: "周杰伦的床边故事"
       alertid: 100002
       belongCD: 8
       cdIdx: 0
       interval: 215
       isonly: 0
       label: "4611686018435776513"
       msgid: 14
      ▶pay: {payalbum: 1, payalbumprice: 2000, paydownload: 1, payinfo: 1, payplay: 0, paytrackmouth: 1,…}
      ▶preview: {trybegin: 65138, tryend: 85421, trysize: 325589}
       rate: 31
      ▶singer: [{id: 4558, mid: "0025NhlN2yWrP4", name: "周杰伦"}]
       size5_1: 0
       size128: 3443771
       size320: 8608939
       sizeape: 24929083
       sizeflac: 24971563
       sizeogg: 5001304
       songid: 107192078
       songmid: "003OUlho2HcRHC"
       songname: "告白气球"
```


图 13-6 歌曲信息

为了简化 URL，在浏览器上进行访问，然后对其请求参数进行删改，对比浏览器返回的响应内容是否和前面一致，最终 URL 优化如下：

```
https://c.y.qq.com/v8/fcg-bin/fcg_v8_singer_track_cp.fcg?loginUin=0&hostUin=0
&singermid=0025NhlN2yWrP4&order=listen&begin=0&num=30&songstatus=1
```

从请求参数分析得知：

（1）singermid 是指歌手的 mid，其作用与 songmid 相同，标识歌手的唯一性。

（2）begin 代表歌曲页数，在网页上单击第二页的时候，会触发相同的请求，发现请求参数 begin 变为 30，说明页数不是按 1、2、3……计算的，而是按照（p-1）*30 的计算公式获取页数的。

（3）num 是每页歌曲数量的间隔数。

综合分析，只要动态设置 singermid 和 begin 的值，就能获取不同歌手的全部歌曲信息。代码如下：

```
    # 获取歌手的全部歌曲
    def get_singer_songs(singermid):
        # 获取歌手姓名和歌曲总数
        url = 'https://c.y.qq.com/v8/fcg-bin/fcg_v8_singer_track_
cp.fcg?
    loginUin=0&hostUin=0&singermid=%s&order=listen&begin=0&num=30&son
gstatus=1'
    %(singermid)
        r = session.get(url)
        # 获取歌手姓名
        song_singer = r.json()['data']['singer_name']
        # 获取歌曲总数
        songcount = r.json()['data']['total']
        # 根据歌曲总数计算总页数
        pagecount = math.ceil(int(songcount) / 30)
        # 循环页数，获取每一页歌曲信息
        for p in range(pagecount):
            url = 'https://c.y.qq.com/v8/fcg-bin/
                   fcg_v8_singer_track_cp.fcg?
                   loginUin=0&hostUin=0&singermid=%s&order=
                   listen&begin=%s&num=30&songstatus=1'
                   %(singermid, p * 30)
            r = session.get(url)
            # 得到每页的歌曲信息
            music_data = r.json()['data']['list']
            # songname：歌名，ablum：专辑，interval：时长，songmid：歌曲
id，用于下载音频文件
            # 将歌曲信息存放于字典 song_dict 中，用于入库
            song_dict = {}
            for i in music_data:
                song_dict['song_name'] = i['musicData']['songname']
                song_dict['song_ablum'] = i['musicData']['albumname']
                song_dict['song_interval'] = i['musicData']
                         ['interval']
                song_dict['song_songmid'] = i['musicData']['songmid']
                song_dict['song_singer'] = song_singer
```

```
        # 下载歌曲
        download(song_dict['song_songmid'])
        # 入库处理，参数 song_dict
        insert_data(song_dict)
        # song_dict 清空处理
        song_dict = {}
```

函数 get_singer_songs() 用于爬取歌手的全部歌曲，代码说明如下：

（1）参数 singermid 代表歌手的唯一值，只需要传入不同歌手的 singermid，就能爬取不同歌手的全部歌曲。

（2）代码有两个相同变量 url，第一个用于动态设置歌手的 singermid，获取歌曲总数和歌手姓名；第二个用于动态设置页数，获取当前歌手每一页的歌曲信息。

（3）下载歌曲调用了 13.2 节实现的 download() 函数；入库处理是调用入库函数 insert_data()，该函数会在后续章节讲解。

13.4 分类歌手列表

现在已实现获取单个歌手的全部歌曲信息，只要在此功能的基础上遍历输入不同歌手的 singermid，就能获取不同歌手的歌曲信息。从 Chrome 开发者工具对歌手列表（y.qq.com/portal/singer_list.html）的分析得知，歌手页数有 5526 页，每页 100 位歌手，全站的歌手共有 552503 位，如图 13-7 所示。

Name	× Headers Preview Response Timing
mod.js?r=2520189	▼GetSingerListCallback({code: 0, data: {list: [,…], per_page: 100, total: 552503, total_page: 5526}, message: "succ",…})
singerlist_7cba3b3.js?max_a...	code: 0
common_dd219c9.js?max_a...	▼data: {list: [,…], per_page: 100, total: 552503, total_page: 5526}
returncode_486a5bc.js?max...	▼list: [,…]
fcg_music_red_dota.fcg?g_t...	▼0: {Farea: "1", Fattribute_3: "3", Fattribute_4: "0", Fgenre: "0", Findex: "X", Fother_name: "Joker",…}
fcg_get_profile_homepage.f...	Farea: "1"
fcg_order_singer_getlist.fcg...	Fattribute_3: "3"
v8.fcg?channel=singer&pa...	Fattribute_4: "0"
emoji.js?max_age=2592000	Fgenre: "0"
pager_6d0a829.js?max_age...	Findex: "X"
gethotkey.fcg?g_tk=354851...	Fother_name: "Joker"
download_5c3a402.js?max_...	Fsinger_id: "5062"
dialog_16f4670.js?max_age...	Fsinger_mid: "002J4UUk29y8BY"
stats?sId=58495963&_=15...	Fsinger_name: "薛之谦"
	Fsinger_tag: "541,555"
	Fsort: "1"
	Ftrend: "0"
	Ftype: "0"
	voc: "0"

图 13-7 歌手信息

从图 13-7 看到，list 是 100 位歌手的信息列表，每条信息的 Fsinger_mid 是歌手的 singermid。如果获取全部歌手的 Fsinger_mid，就需要循环 552503 次，根据项目设计，将循环次数按字母分类划分。

在歌手列表页上使用字母 A ～ Z 对歌手进行分类筛选，利用这个分类功能可以将全部歌手分为两层循环：第一层是循环每一个字母分类，第二层是循环每个分类下的总页数，拆分两层循环主要为异步编程提供切入点，具体实现方式会在后面的章节讲解。

在网页上单击分类“A”，可在开发者工具的 JS 标签看到相应的请求信息，如图 13-8 所示。

Name
v8.fcg?channel=singer&pa...
1 / 12 requests | 4.1 KB / 4.3 K...
× Headers Preview Response Cookies Timing
▼GetSingerListCallback({code: 0, data: {list: [,…], per_page: 100, total: 39707, total_page: 398}, message: "succ"
code: 0
▼data: {list: [,…], per_page: 100, total: 39707, total_page: 398}
▼list: [,…]
▼0: {Farea: "3", Fattribute_3: "0", Fattribute_4: "0", Fgenre: "0", Findex: "A", Fother_name: "艾伦·沃克",…}
Farea: "3"
Fattribute_3: "0"
Fattribute_4: "0"
Fgenre: "0"
Findex: "A"
Fother_name: "艾伦·沃克"
Fsinger_id: "944940"
Fsinger_mid: "0020PeOh4ZaCw1"
Fsinger_name: "Alan Walker (艾伦·沃克)"
Fsinger_tag: ""
Fsort: "6"
Ftrend: "1"
Ftype: "0"
voc: "0"
▶1: {Farea: "0", Fattribute_3: "2", Fattribute_4: "0", Fgenre: "0", Findex: "A", Fother_name: "黄丽玲",…}
▶2: {Farea: "3", Fattribute_3: "7", Fattribute_4: "0", Fgenre: "0", Findex: "A", Fother_name: "阿黛尔",…}

图 13-8　歌手分类列表

分析请求链接并以代码形式展现：

```
    https://c.y.qq.com/v8/fcg-bin/v8.fcg?channel=singer&page=list&key
=all_all_A&pagesize=100
    &pagenum=1&g_tk=354851523&jsonpCallback=GetSingerListCallback&log
inUin=554301449&hostUin=0
    &format=jsonp&inCharset=utf8&outCharset=utf-8&notice=0&platform=y
qq&needNewCode=0
```

按照前面的处理方式，对 URL 进行删减优化处理，最终得到的 URL 如下：

```
    https://c.y.qq.com/v8/fcg-bin/v8.fcg?channel=singer&page=list&key
=all_all_A&pagesize=100
    &pagenum=1&loginUin=0&hostUin=0&format=jsonp
```

从 URL 的请求参数分析得知：

（1）key 代表筛选条件，all_all_A 代表字母分类 A。

（2）pagenum 代表页数，页数按 1、2、3……方式计算。

（3）pagesize=100 代表每页有 100 位歌手，该参数的值固定不变。

（4）loginUin、hostUin 和 format 参数固定不变。

综合上述分析，功能代码如下：

```
# 获取当前字母下的全部歌手
def get_genre_singer(key, page_list):
    # 遍历当前字母分类的总页数
    for p in page_list:
        url = 'https://c.y.qq.com/v8/fcg-bin/v8.fcg?channel=
              singer&page=list&key=all_all_%s'
&pagesize=100&pagenum=%s&loginUin=0&hostUin=0&format=jsonp'
             % (key, p + 1)
        r = session.get(url)
        # 遍历每一页的每一个歌手
        for k in r.json()['data']['list']:
            singermid = k['Fsinger_mid']
            # 得到的 singermid 传入 13.3 节实现的函数方法
            get_singer_songs(singermid)
```

函数 get_genre_singer() 用于爬取当前字母分类下全部歌手的歌曲信息：

（1）参数 key 和 page_list 分别是当前分类和当前分类的歌手总页数（列表结构）。

（2）外层循环用于遍历当前分类的总页数。

（3）内层循环用于遍历当前分类每页每位歌手的 singermid，并调用函数 get_singer_songs() 获取每一位歌手的全部歌曲。

13.5 全站歌手列表

13.4 节已实现爬取单个字母分类的全部歌手的歌曲信息。如果想获取全站的歌曲

信息，那么只需要在 13.4 节实现的功能上再遍历 26 个字母，实现代码如下：

```
# 获取全站歌手
def get_all_singer():
    # 获取字母 A ～ Z 的全站歌手
    for i in range(65, 90):
        # 通过 ASCII 转换字母
        key = chr(i)
        # 获取当前字母分类的总歌手页数，并转换列表结构
        url = 'https://c.y.qq.com/v8/fcg-bin/v8.fcg?channel=
               singer&page=list&key=all_all_%s
&pagesize=100&pagenum=%s&loginUin=0&hostUin=0&format=
                      jsonp' % (key, 1)
        r = session.get(url, headers=headers)
        pagenum = r.json()['data']['total_page']
        # 构建参数
        page_list = [x for x in range(pagenum)]
        # 调用 13.4 节实现的函数
        get_genre_singer(key, page_list)
# 主程序运行
if __name__ == '__main__':
    get_all_singer()
```

上述代码功能说明如下：

（1）循环从 65 开始，到 90 结束，65 ～ 90 在 ASCII 中代表字母 A ～ Z。

（2）将当前循环次数 i 转换成字母，并赋值给 key 变量。

（3）向网站发送请求，获取每个字母分类的歌手总页数。

（4）将总页数生成列表结构，作为函数 get_genre_singer() 的参数。

上述代码是整个项目程序的运行入口，程序运行执行函数的顺序如下：

（1）get_all_singer()：循环 26 个字母，构建参数并调用函数 get_genre_singer()。

（2）get_genre_singer(key, page_list)：遍历当前分类总页数，获取每页每位歌手的歌曲信息。

（3）get_singer_songs(singermid)：实现歌手的歌曲入库和下载。

（4）download(songmid)：下载歌曲。

（5）insert_data(song_dict)：入库处理。

13.6 数据存储

在逻辑功能实现过程中发现数据入库的函数 insert_data()，该函数主要存放在 music_db.py 中，本节使用 SQLAlchemy 实现数据入库。

根据爬虫规则分析，入库的数据有歌名、所属专辑、时长、歌曲 mid（下载歌曲文件以歌曲 mid 命名）和歌手姓名。针对所爬取的数据及性质，数据库命名如表 13-1 所示。

表13-1 song数据表

字 段	中文名
song_id	主键
song_name	歌名
song_ablum	所属专辑
song_interval	时长
song_songmid	歌曲mid
song_singer	歌手姓名

SQLAlchemy 映射数据库代码如下：

```
from sqlalchemy import *
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base
# 连接数据库
engine=create_engine("mysql+pymysql://root:1234@localhost:3306/
music_db?charset=utf8")
# 创建会话对象，用于数据表的操作
DBSession = sessionmaker(bind=engine)
SQLsession = DBSession()
Base = declarative_base()
# 映射数据表
class song(Base):
```

```
        # 表名
        __tablename__ ='song'
        # 字段、属性
        song_id = Column(Integer, primary_key=True)
        song_name = Column(String(50))
        song_ablum = Column(String(50))
        song_interval = Column(String(50))
        song_songmid = Column(String(50))
        song_singer = Column(String(50))
    # 创建数据表
    Base.metadata.create_all(engine)
```

完成 SQLAlchemy 和数据库的映射，在上述代码中补充 insert_data() 函数，代码如下：

```
    def insert_data(song_dict):
        # 连接数据库
        engine = create_engine("mysql+pymysql://root:1234@localhost:
                 3306/music_db?charset=utf8")
        # 创建会话对象，用于数据表的操作
        DBSession = sessionmaker(bind=engine)
        SQLsession = DBSession()
        data = song(
            song_name = song_dict['song_name'],
            song_ablum = song_dict['song_ablum'],
            song_interval = song_dict['song_interval'],
            song_songmid = song_dict['song_songmid'],
            song_singer = song_dict['song_singer'],
        )
        SQLsession.add(data)
        SQLsession.commit()
```

函数 insert_data() 主要对传递的参数 song_dict 进行入库处理，参数 song_dict 为字典格式。函数运行会创建新的数据库连接，创建新数据库连接主要是为异步编程做准备。

上述代码存放在 music_db.py 文件中，在 music.py 中只需导入 music_db.py 的 insert_data() 函数即可实现数据入库。

13.7 分布式概念

爬虫的爬取效率是实际生产中一个重要的考虑因素，时间就是金钱，更是一个企业能够生存下来的准则之一。为了提高爬虫的效率，在此为大家介绍异步编程开发思想，简单地说，就是利用多进程和多线程实现爬虫开发。

GIL 和多线程的关系值得注意。很多读者会对 Python 的多线程有一定的误解，Python 执行环境大部分依赖于 GIL，而 GIL 限制了多线程的功能。

13.7.1 GIL 是什么

首先需要明确的一点是，GIL 并不是 Python 的特性，它是在实现 Python 解析器（CPython）时所引入的一个概念。就好比 C++ 是一套语言（语法）标准，但是可以用不同的编译器来编译成可执行代码。有名的编译器有 GCC、INTEL C++，Visual C++ 等。Python 也一样，同样的代码可以通过 CPython、PyPy、Psyco 等不同的 Python 执行环境来执行。像其中的 JPython 就没有 GIL。然而因为 CPython 是大部分环境下默认的 Python 执行环境，所以在很多人的概念里 CPython 就是 Python，也就想当然地把 GIL 归结为 Python 语言的缺陷。这里要先明确一点：GIL 并不是 Python 的特性，Python 完全可以不依赖于 GIL。

13.7.2 为什么会有 GIL

由于物理上的限制，各个 CPU 厂商在核心频率上的比赛已经被多核所取代。为了更有效地利用多核处理器的性能，就出现了多线程的编程方式，而随之带来的就是线程间数据一致性和状态同步的困难。

为了利用多核，Python 开始支持多线程。而解决多线程之间数据完整性和状态同步的最简单的方法就是加锁。 于是有了 GIL 这把超级大锁，而当越来越多的代码库开发者接受了这种设定后，他们开始大量依赖这种特性（即默认 Python 内部对象是 thread-safe 的，无须在实现时考虑额外的内存锁和同步操作）。

Python 在设计之初就考虑到要在解释器的主循环中同时只有一个线程在执行，即在任意时刻，只有一个线程在解释器中运行。对 Python 虚拟机的访问由全局解释器锁（GIL）来控制，正是这个锁能保证同一时刻只有一个线程在运行。

在多线程环境中，Python 解释器按以下方式执行：

（1）设置 GIL。

（2）切换到一个线程去运行。

（3）运行：指定数量的字节码指令或者线程主动让出控制（可以调用 time.sleep(0)）。

（4）把线程设置为睡眠状态。

（5）解锁 GIL。

（6）再次重复以上所有步骤。

有人认为 Python 的多线程比较“鸡肋”，这种说法只是相对而言的，Python 是仅有的支持多线程的解释型语言（Perl 的多线程是残疾的，PHP 没有多线程）。相对自身而言，如果代码是 CPU 密集型的，并且是线性执行，在这种情况下多线程就是“鸡肋”，效率可能还不如单线程；如果代码是 IO 密集型的，多线程可以明显提高效率，例如爬虫，在绝大多数时间都在等待服务器返回数据和频繁的数据读写。

13.8 并发库 concurrent.futures

Python 标准库为我们提供了 threading 和 multiprocessing 模块编写相应的多线程 / 多进程代码。从 Python 3.2 开始，标准库为我们提供了 concurrent.futures 模块，它提供了 ThreadPoolExecutor 和 ProcessPoolExecutor 两个类，实现了对 threading 和 multiprocessing 更高级的抽象，对编写线程池 / 进程池提供了直接的支持。

下面通过简单的例子讲解如何使用 concurrent.futures，代码如下：

```
# 导入 concurrent.futures 模块
from concurrent.futures import ThreadPoolExecutor,
ProcessPoolExecutor
import datetime

# 线程的执行方法
def print_value(value):
    print('Thread' + str(value))
```

```
# 每个进程里面的线程
def myThread(value):
    Thread = ThreadPoolExecutor(max_workers=2)
    Thread.submit(print_value, datetime.datetime.now())
    Thread.submit(print_value, datetime.datetime.now())

# 创建两个进程，每个进程执行 myThread 方法，myThread 主要将每个进程通过线程执行
# 如果不填写 max_workers=2，就会根据计算机的每一个 CPU 创建一个 Python 进程，如果四核就创建 4 个进程
def myProcess():
    pool = ProcessPoolExecutor(max_workers=2)
    pool.submit(myThread, datetime.datetime.now())
    pool.submit(myThread, datetime.datetime.now())

if __name__ == '__main__':
    myProcess()
```

在上述代码中，创建了进程 ProcessPoolExecutor 和线程 ThreadPoolExecutor，其中在每个进程中又创建了两个线程。

下面简单讲述一下 concurrent.futures 属性和方法。

- Executor：Executor 是一个抽象类，它不能被直接使用。为具体的异步执行定义了基本的方法：ThreadPoolExecutor 和 ProcessPoolExecutor 继承了 Executor，分别被用来创建线程池和进程池的代码。
- 创建进程和线程之后，Executor 提供了 submit() 和 map() 方法对其操作。submit() 和 map() 最大的区别是参数类型，map() 的参数必须是列表、元组和迭代器的数据类型。
- Future：可以理解为一个在未来完成的操作，这是异步编程的基础。通常情况下，我们执行 IO 操作和访问 URL 时，在等待结果返回之前会产生阻塞，CPU 不能做其他事情，而 Future 的引入帮助我们在等待的这段时间可以完成其他的操作。

13.9 分布式爬虫

我们已经知道，爬取全站歌曲信息是按照字母 A ～ Z 依次循环爬取的，这是在单进程单线程的情况下运行的。如果将这 26 次循环分为 26 个进程同时执行，每个进程只需执行对应的字母分类，假设执行一个分类的时间相同，那么多进程并发的效率是单进程的 26 倍。

除了运用多进程之外，项目代码大部分是 IO 密集型的，那么在每个进程下使用多线程也可以提高每个进程的运行效率。我们知道歌手列表页是通过两层循环实现的，第一层是循环每个分类字母，现将每个分类字母作为一个单独进程处理；第二层是循环每个分类的歌手总页数，可将这个循环使用多线程处理。假设每个进程使用 10 条线程（线程数可自行设定，具体看实际需求），那么每个进程的效率也相对提高 10 倍。

分布式策略考虑的因素有网站服务器负载量、网速快慢、硬件配置和数据库最大连接量。举个例子，爬取某个网站 1000 万数据，从数据量分析，当然进程和线程越多，爬取的速度越快。但往往忽略了网站服务器的并发量，假设设定 10 个进程，每个进程 200 条线程，每秒并发量为 200×10=2000，若网站服务器并发量远远低于该并发量，在请求网站的时候，就会出现卡死的情况，导致请求超时（即使对超时做了相应处理），无形之中增加等待时间。除此之外，进程和线程越多，对运行程序的系统的压力越大，若涉及数据入库，还要考虑并发数是否超出数据库连接数。

根据上述分布式策略，在 music_db.py 中添加代码如下：

```
def myThread(genre):
    # 每个字母分类的歌手列表页数
    url = 'https://c.y.qq.com/v8/fcg-bin/v8.fcg?channel=
           singer&page=list&
key=all_all_%s&pagesize=100&pagenum=%s&loginUin=0&hostUin=
                        0&format=jsonp' %(genre, 1)
    r = session.get(url, headers=headers)
    pagenum = r.json()['data']['total_page']
    page_list = [x for x in range(pagenum)]
    # 设置线程数
    thread_number = 10
    # 将每个分类总页数平均分给线程数
    list_interval = math.ceil(len(page_list) / thread_number)
```

```
    # 设置线程对象
    Thread = ThreadPoolExecutor(max_workers=thread_number)
    for index in range(thread_number):
        # 计算每条线程应执行的页数
        start_num = list_interval * index
        if list_interval * (index + 1) <= len(page_list):
            end_num = list_interval * (index + 1)
else:
end_num = len(page_list)
        # 每个线程各自执行不同的歌手列表页数
        Thread.submit(get_genre_singer, genre,
                      page_list[start_num: end_num])
# 多进程
def myProcess():
    with ProcessPoolExecutor(max_workers=26) as executor:
        for i in range(65, 90):
            # 创建 26 个进程，分别执行 A ～ Z 分类
            executor.submit(myThread, chr(i))
# 主程序运行
if __name__ == '__main__':
    myProcess()
```

代码中定义了 myProcess() 和 myThread() 方法函数，分别实现多进程和多线程。

- 多进程 myProcess() 函数：主要是循环字母 A ～ Z，将每个字母独立创建一个进程，每个进程执行的方法函数是 myThread()，参数是当前的分类字母。
- 多线程 myThread() 函数：首先根据传入参数获取当前分类的歌手总页数，然后根据得到的总页数和设定的线程数计算每条线程应执行的页数，最后遍历设定线程数，让每条线程执行相应的页数。例如总页数 100 页，10 条线程，每条线程应执行 10 页，第一条线程执行 0 ～ 10 页，第二条线程执行 10 ～ 20 页，以此类推。线程调用的方法函数是 get_genre_singer()，该方法函数是 13.4 节实现的功能。

在实现分布式爬虫的时候，必须注意的是：

（1）全局变量不能放在 if __name__ == '__main__' 中，因为使用多进程的时候，新开的进程不会在此获取数据。

（2）使用 SQLalchemy 入库最好重新创建一个数据库连接，如果多个线程和进程共同使用一个连接，就会出现异常。

（3）分布式策略最好在程序代码的最外层实现。例如在项目中，get_singer_songs() 方法函数里有两个循环，不建议在此使用分布式处理，在代码底层实现分布式不是不可行，只是代码变动太大，而且考虑的因素较多，代码维护相对较难。

13.10 本章小结

本章以 QQ 音乐为爬取对象，爬取范围是全站的歌曲信息，爬取方式在歌手列表获取每一位歌手的全部歌曲。如果爬取的数量较大，就使用异步编程实现分布式爬虫开发，可提高爬虫效率。读者应重点掌握以下内容：

1. 项目实现的功能

（1）歌曲下载 download(songmid)：爬虫最底层的功能，也是爬虫最核心的功能。

（2）歌手和歌曲信息 get_singer_songs(singermid)：将歌手的歌曲信息入库和歌曲下载。

（3）分类歌手列表 get_genre_singer(key, page_list)：获取单一字母分类的全部歌手和歌曲信息。

（4）全站歌手列表 get_all_singer()：获取全站歌手和歌曲信息。

（5）数据存储 insert_data(song_dict)：将爬取的歌手和歌曲信息入库处理。

（6）多进程 myProcess()：每个字母分类创建一个单独进程运行。

（7）多线程 myThread(genre)：每个进程使用多线程爬取数据。

2. 分布式策略考虑的因素

分布式策略考虑的因素有网站服务器负载量、网速快慢、硬件配置和数据库最大连接量。举个例子，比如爬取某个网站 1000 万数据，从数据量分析，当然进程和线程越多，爬取的速度越快。但往往忽略了网站服务器的并发量，假设设定 10 个进程，每个进程 200 条线程，每秒并发量为 200×10=2000，若网站服务器并发量远远低于该并发量，在请求网站的时候，就会出现卡死的情况，导致请求超时（即使对超时做了相应处理），无形之中增加等待时间。除此之外，进程和线程越多，对程序运行的系统的压力越大，若涉及数据入库，还要考虑并发数是否超出数据库连接数。

3. 实现分布式爬虫的注意事项

（1）全局变量不能放在 if __name__ == '__main__' 中，因为使用多进程的时候，新开的进程不会在此获取数据。

（2）使用 SQLalchemy 入库最好重新创建一个数据库连接，如果多个线程和进程共同使用一个连接，就会抛出异常。

（3）分布式策略最好在程序代码的最外层实现。例如在项目中，get_singer_songs() 方法函数里有两个循环，不建议在此使用分布式处理，在代码底层实现分布式不是不可行，只是代码变动太大，而且考虑的因素较多，代码维护相对较难。



本章项目仅用于爬虫教学，并无违反版权法规之意，请读者注意自觉树立版权保护意识。

第 14 章

项目实战：爬虫软件——淘宝商品信息

14.1 分析说明

对于一个完整的爬虫项目来说，完成功能开发仅仅是完成了主要功能，不可能直接将代码交付给客户，让客户自己运行，而且源代码没有封装处理，很容易遭到修改和破坏。为了提高用户体验和保护源代码的完整，大多数爬虫主要以软件的形式交付给客户使用。其中最为典型的是抢票软件，这类软件的原理是在爬虫的基础上以软件为载体供用户使用。本项目在第 12 章的基础上进一步完善和扩展，主要讲述如何以软件形式实现淘宝商品信息开发。

14.2 GUI 库介绍

Python 提供了多个图形开发界面的库，常用的 GUI 库有：

- Tkinter（也叫 Tk 接口）是 Tk 图形用户界面工具包标准的 Python 接口。Tk 是一个轻量级的跨平台图形用户界面（GUI）开发工具，可以运行在大多数 UNIX 平台、Windows 系统和 Mac 系统中。
- wxPython 是 Python 语言的一套优秀的 GUI 图形库，允许 Python 程序员很方便地创建完整的、功能键全的 GUI 用户界面。wxPython 是作为优秀的跨平台 GUI 库，以 wxWidgets 的 Python 封装和 Python 模块的方式提供给用户。
- PyQt 是 Qt 库的 Python 版本。PyQt3 支持 Qt1 到 Qt3，PyQt4 支持 Qt4，PyQt5 支持 Qt5。PyQt 的首次发布是在 1998 年，当时叫作 PyKDE，因为那时 SIP 和 PyQt 没有分开。PyQt 是用 SIP 写的，提供 GPL 版和商业版。
- Kivy 是一个开源工具包，是能够使用相同源代码创建的程序，并且可以跨平台运行。它主要关注创新型用户界面开发，如多点触摸应用程序。Kivy 还提供一个多点触摸鼠标模拟器。Kivy 当前支持的平台包括 Linux、Windows、Mac 和 Android，拥有能够处理动画、缓存、手势和绘图等功能。Kivy 还内置许多用户界面控件，如按钮、摄影机、表格、Slider 和树形控件等。
- Flexx 是一个纯 Python 工具包，用来创建图形化界面应用程序，使用 Web 技术进行界面的渲染。可以用 Flexx 来创建桌面应用，同时也可以导出一个应用到独立的 HTML 文档。因为使用纯 Python 开发，所以 Flexx 是跨平台使用的。只需要有 Python 和浏览器，Flexx 就可以运行。

在本项目中，主要讲解如何使用 PyQt5 实现软件开发。

14.3 PyQt5 安装及环境搭建

PyQt5 是一套绑定 Qt5 的应用程序框架，由 Python 语言实现，已经有超过 620 个类和 6000 个函数与方法。PyQt5 是一个运行在所有主流操作系统上的多平台组件，包括 UNIX、Windows 和 Mac OS。PyQt5 是双重许可的，开发者可以选择 GPL 和商业许可。

PyQt5 可以使用 pip 安装：

```
pip install PyQt5
```

完成 PyQt5 的安装后，接着安装图形界面的开发工具，这是能快速开发图形界面的辅助工具。如果对 PyQt5 比较熟悉，可以使用 Python 纯代码开发图形界面。

开发工具有 Qt Creator 与 Qt Designer，两者都能实现图形界面的开发。其中，后者是前者一部分功能的"阉割"。Qt Creator 包括项目生成向导、高级的 C++ 代码编辑器、浏览文件及类的工具，集成了 Qt Designer、Qt Assistant、Qt Linguist、图形化的 GDB 调试前端和 qmake 构建工具等。

安装 Qt Creator 可以到官方网站下载 .exe 安装包（https://www1.qt.io/download/），下载安装包前需要注册和填写个人信息才行。

Qt Designer 仅支持在 Windows 安装，并且可以使用 pip 安装：

```
pip install PyQt5-tools
```

本书以 Qt Designer 作为图形界面开发工具，安装 Qt Designer 后，可以在 Python 安装目录 \Lib\site-packages\pyqt5-tools 找到 designer.exe，双击并打开 Qt Designer，如图 14-1 所示。

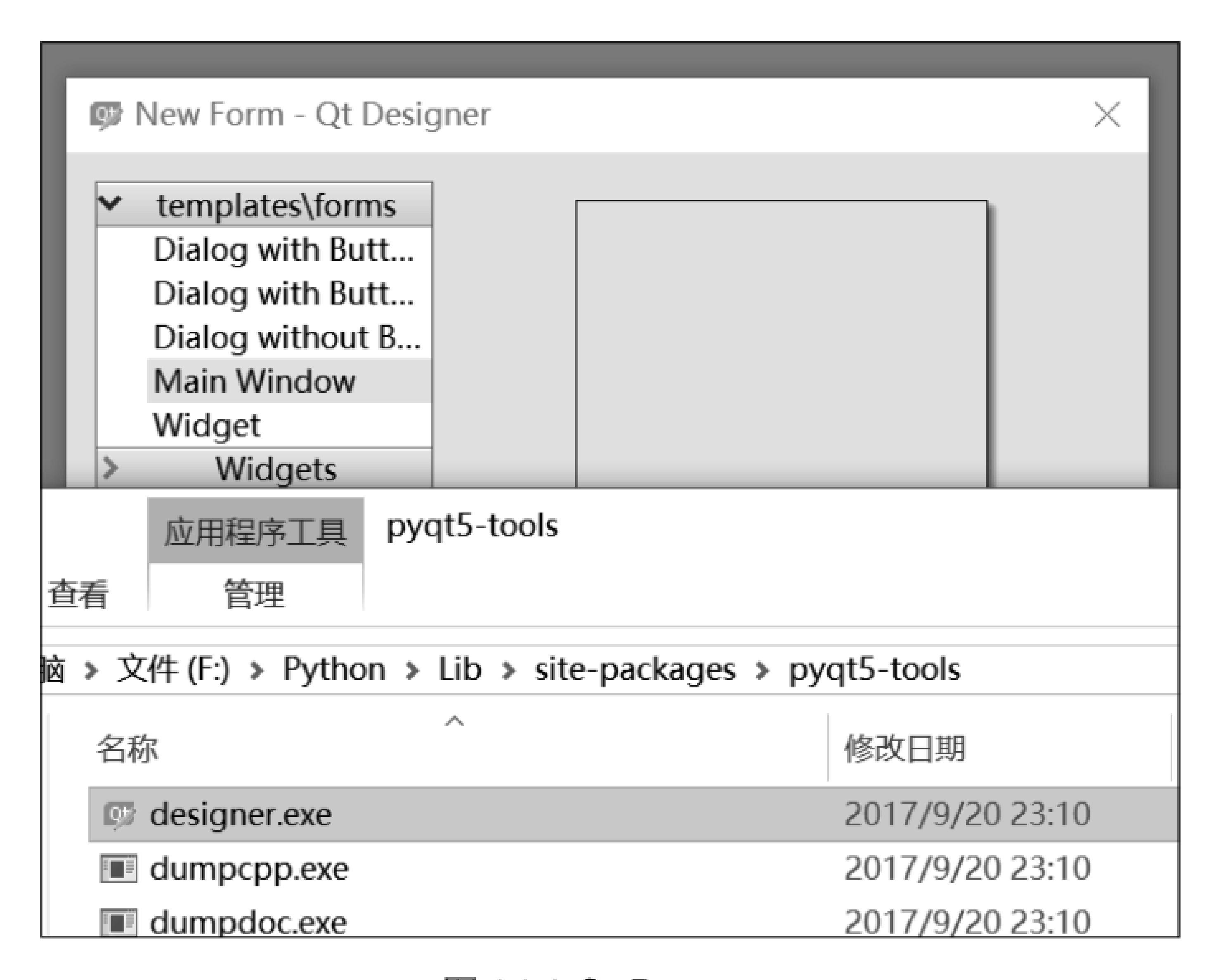


图 14-1 Qt Designer

安装 PyQt5 和 Qt Designer（Qt Creator）之后，接下来在 PyCharm 搭建开发环境。为什么要在 PyChram 搭建开发环境，由于我们使用 Qt Designer（Qt Creator）创建并生成图形界面文件，文件以 ui 为后缀名，在 Python 中无法识别该文件内容，搭建环境的目的是将 ui 文件转换成 py 文件。

不同的 PyChram 版本配置的步骤有所区别，以 Windows 的 PyCharm 为例，PyChram 版本如图 14-2 所示。

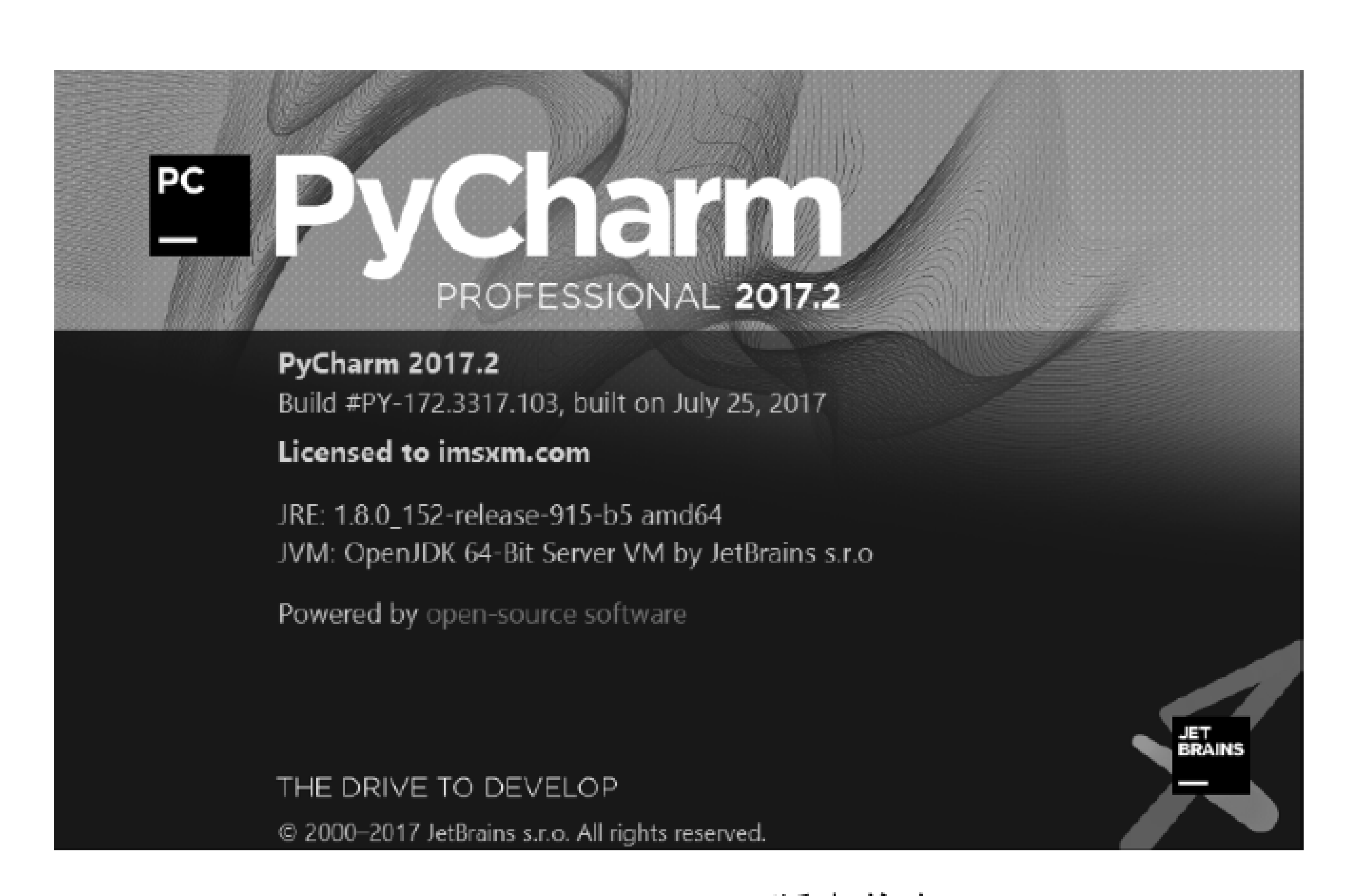


图 14-2 PyChram 版本信息

配置步骤如下：

步骤 01 单击“File”里面的“Settings”，找到“Tools”里面的“External Tools”，如图 14-3 所示。

步骤 02 单击“Tools→External Tools”下方的“+”，新建一个 Tool，输入信息，如图 14-4 所示。

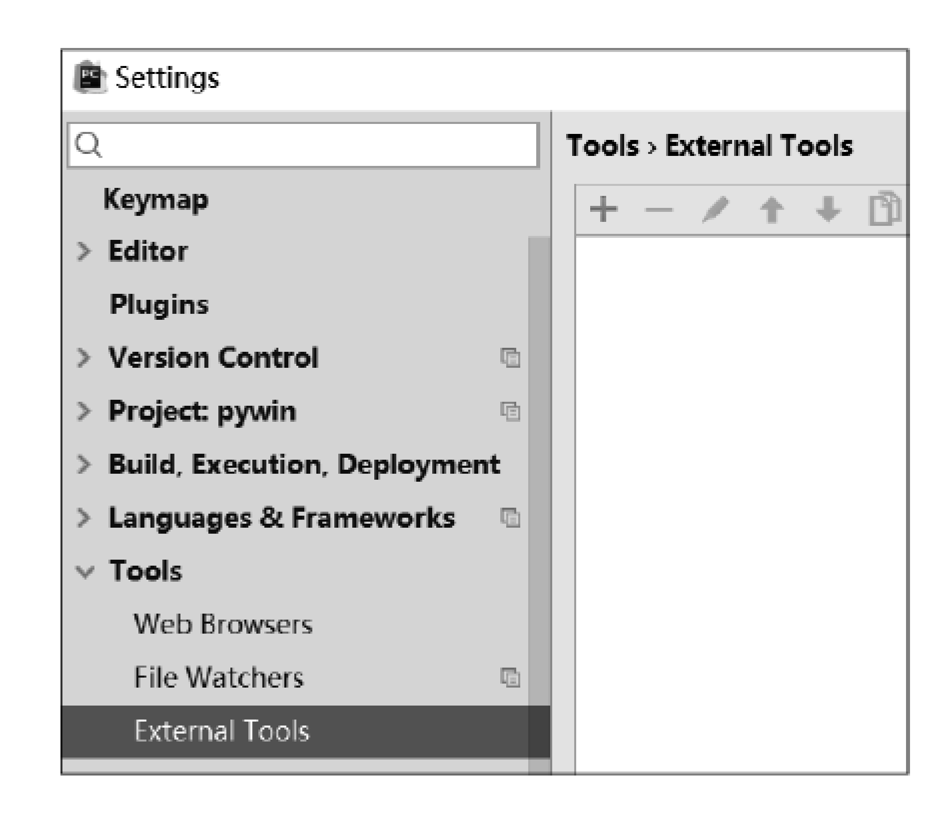


图 14-3 External Tools

从图 14-4 中看到，Program 的内容是 Python 安装目录的 python.exe，这是 Python 解释器；Parameters 是将 ui 文件转换为 py 文件的命令行；Working directory 是转换后生成文件的保存路径。

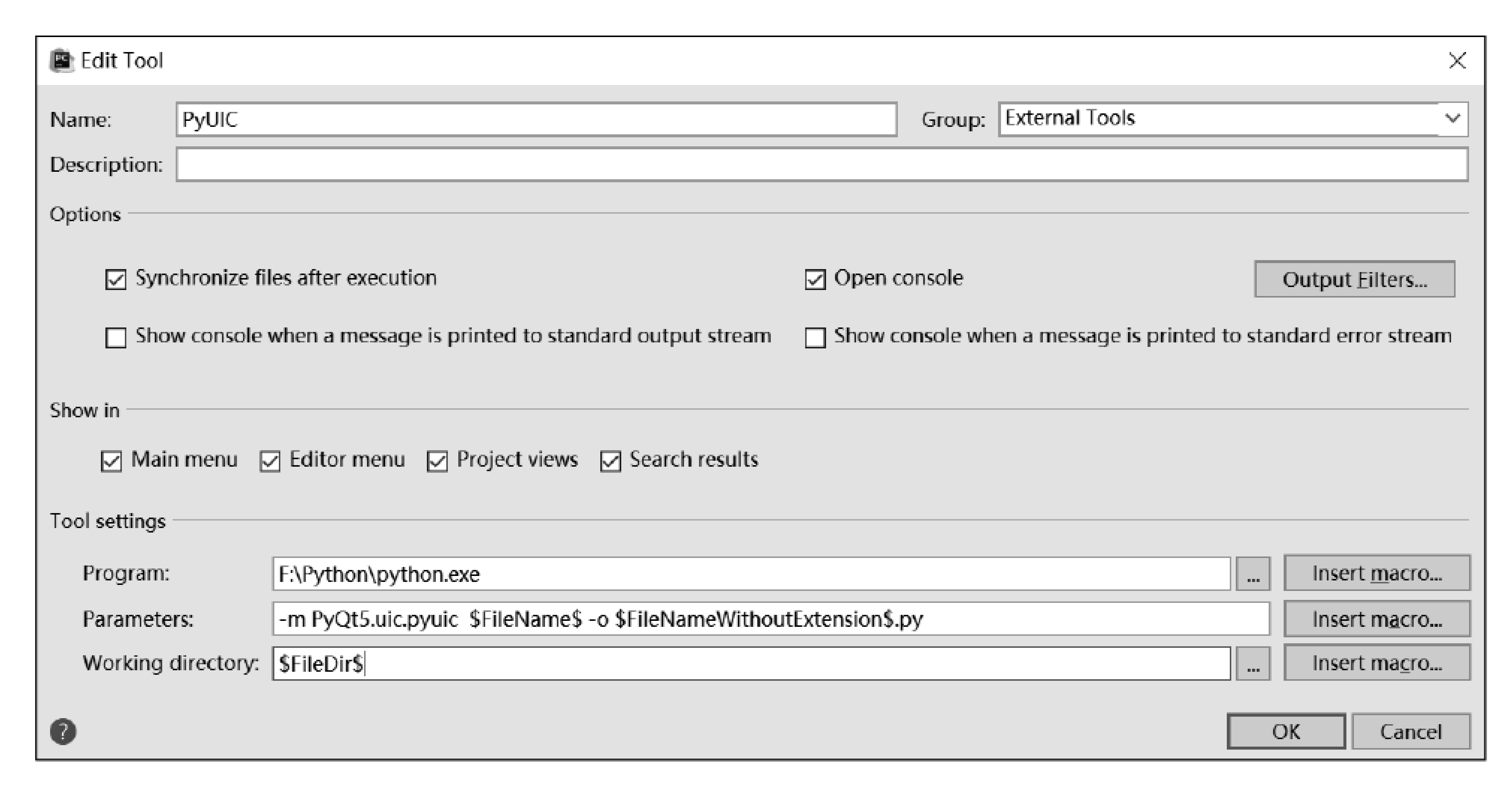


图 14-4 配置 Tools

配置 PyChram 主要是将 ui 文件快速转换成 py 文件，不是一定要配置在 PyChram 才能转换文件，也可以在 CMD（终端）界面运行 Parameters 中的命令行来实现转换。

完成了 PyQt5 和 Qt Designer（Qt Creator）的安装，并在 PyChram 配置了文件转换工具，接下来开始讲解软件开发。

14.4 软件界面开发

软件界面开发由 Qt Designer 完成，打开 Qt Designer，可以看到一个新建窗口的界面，有 5 种功能模板可供选择，如图 14-5 所示。

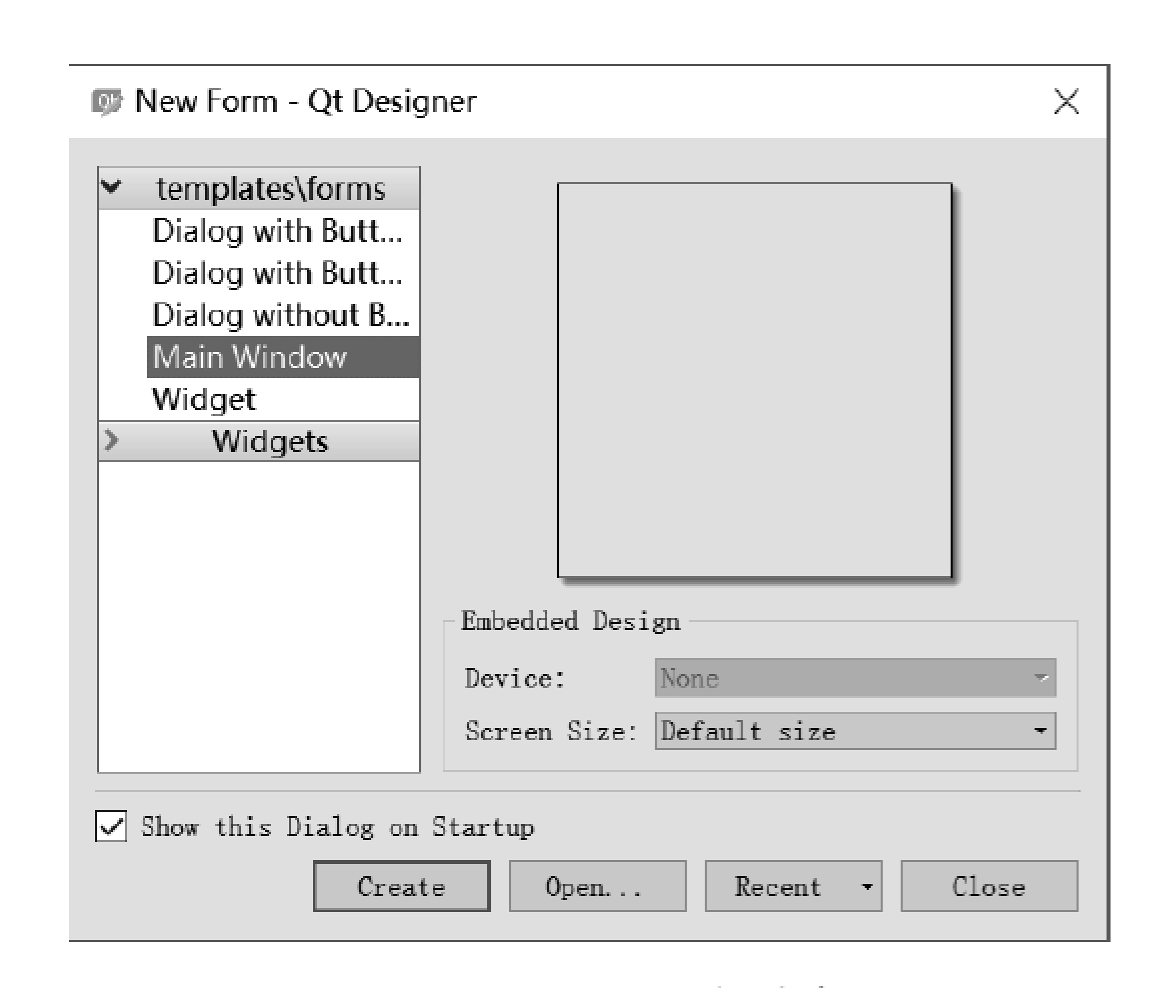


图 14-5 Qt Designer 新建窗口

虽然新建窗口提供 5 种功能模板，但实际上只有 3 种不同类型的模板，分别是 Dialog、MainWindow 和 Widget，三者作用如下：

（1）MainWindow 是主界面，一个窗口是父 / 子 hiearchy 的顶部，通常显示标题栏和边框。底层窗口系统（Windows、KDE、GNOME 等）将为窗口提供策略，如标题栏 / 边框样式、布局和焦点等。

（2）Widget 是小部件，是屏幕上的一个矩形区域，用于显示和用户交互，包括按钮、滑块、视图、对话框和窗口等。所有窗口小部件将在屏幕上显示某些内容，许多窗口小部件也将接受来自键盘或鼠标的用户输入。“widget”一词来自 UNIX，在 Windows 中称为“控件”。

（3）Dialog 为对话框，通常是临时的，可以设置不同的标题栏外观，主要用于通知或收集输入窗口，并且底部或右侧通常具有 OK、Cancel 等按钮。

在此，选择并进入 Dialog 模板，如图 14-6 所示。

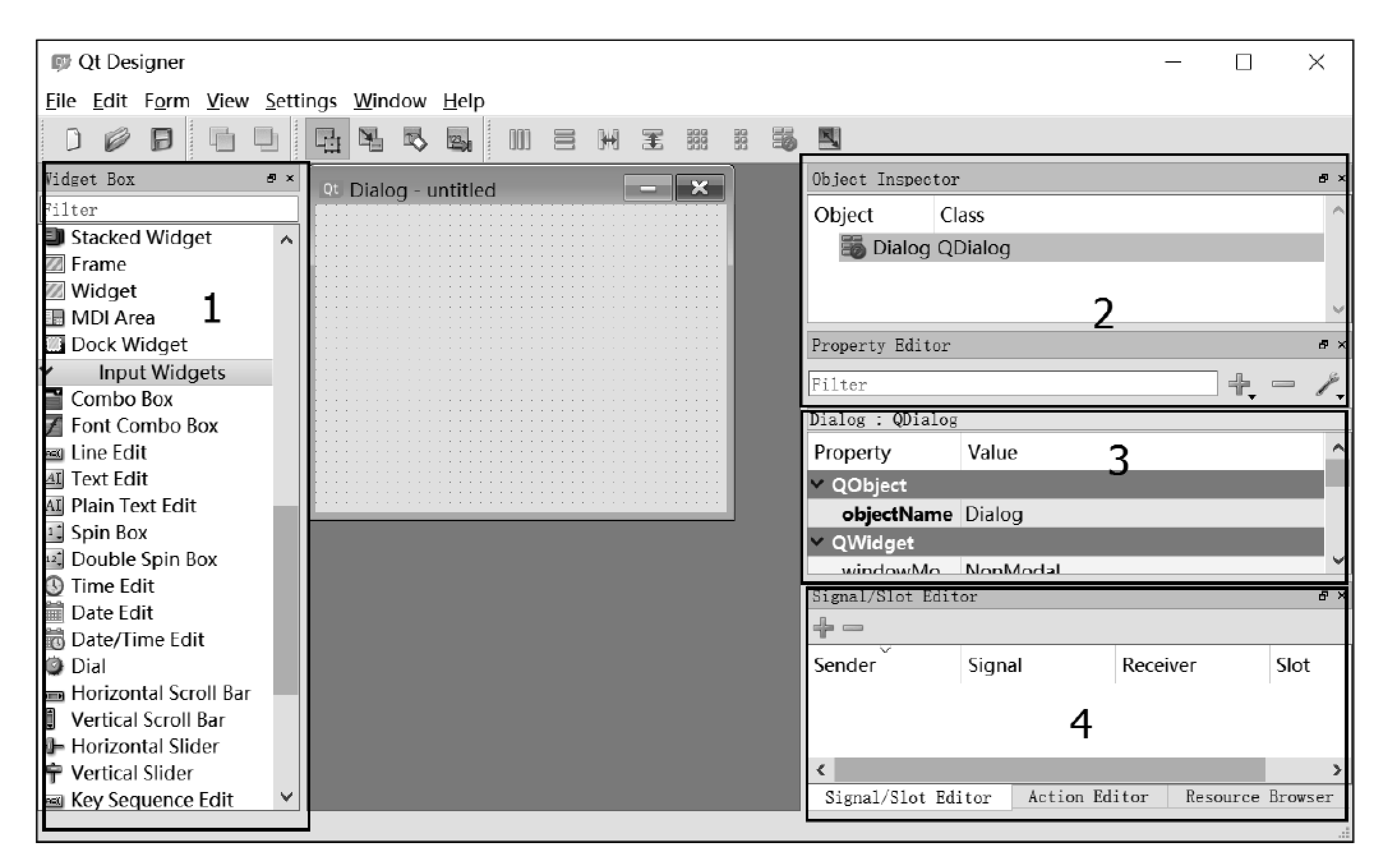


图 14-6 Qt Designer 设计软件界面

从图 14-6 中可以看到，除了正中间区域是软件设计的界面之外，还有为以下 4 个区域。

- 区域 1：控件区，软件的功能控件都在此区域生成，可以拖来控件到模板上实现可视化软件设计。

- 区域 2：软件的目录结构，显示模板中所有控件的类型，能帮助设计者快速找到控件。
- 区域 3：控件属性区，主要修改控件的属性。
- 区域 4：信号（Signal）/ 槽（Slot），信号和槽是 Qt 编程中对象间的通信机制。简单地说，就是单击按钮时候所触发的事件。单击按钮称之为信号，触发的事件称之为槽。

接下来开始设计爬虫软件界面。

通过第 12 章可知，淘宝商品爬虫可以根据用户的需求设置关键字和爬取页数。因此，爬虫软件根据这两个设置条件进行开发设计，如图 14-7 所示。

采集页数用的是 ComboBox 下拉框控件，关键字是 Plain Text Edit 文本框控件，采集按钮是 Push Button 控件，保存文件名为 taobao_v.ui，文件路径为 F:\TB\taobao_v.ui。

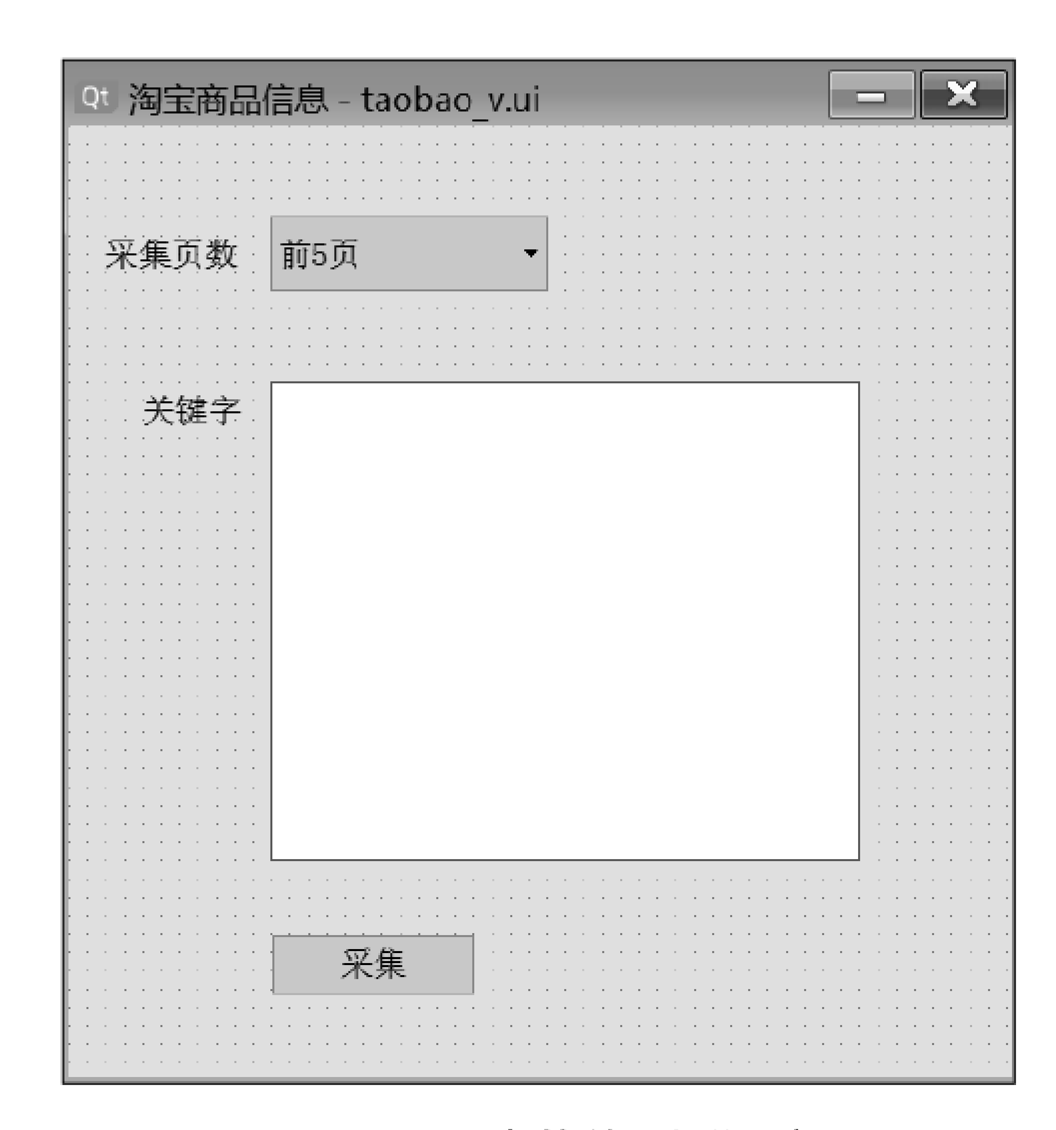


图 14-7　爬虫软件界面设计

下一步是将 ui 文件转换成 py 文件，在 PyChram 中打开 F:\TB，TB 文件夹里只有 taobao_v.ui 文件，选中 PyChram 里的 taobao_v.ui 文件并右击，找到 External Tools，单击 PyUIC。

代码运行完成之后，在同一目录下可以看到其自动生成的 taobao_v.py 文件，如图 14-8 所示。

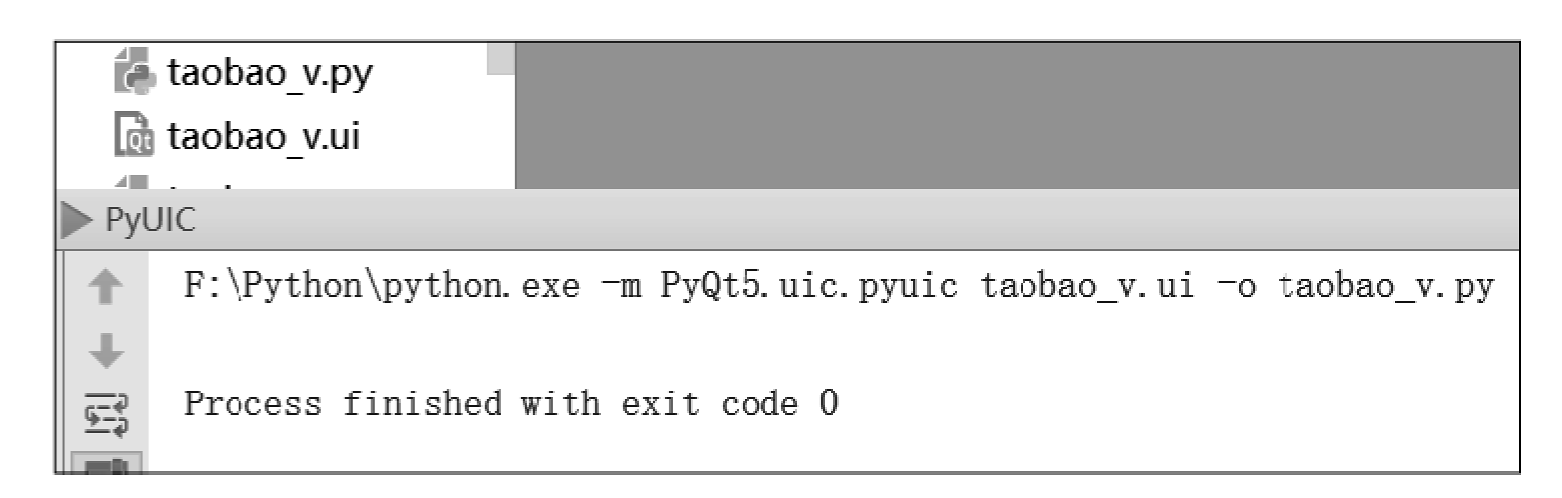


图 14-8　ui 文件转换为 py 文件

查看 taobao_v.py，代码如下：

```
from PyQt5 import QtCore, QtGui, QtWidgets
class Ui_Dialog(object):
    def setupUi(self, Dialog):
        Dialog.setObjectName("Dialog")
        Dialog.resize(513, 518)
        self.pushButton = QtWidgets.QPushButton(Dialog)
        self.pushButton.setGeometry(QtCore.QRect(110, 440, 112, 34))
        self.pushButton.setObjectName("pushButton")
        self.plainTextEdit = QtWidgets.QPlainTextEdit(Dialog)
        self.plainTextEdit.setGeometry(QtCore.QRect(110, 140, 321, 261))
        self.plainTextEdit.setObjectName("plainTextEdit")
        self.label = QtWidgets.QLabel(Dialog)
        self.label.setGeometry(QtCore.QRect(40, 140, 61, 31))
        self.label.setObjectName("label")
        self.comboBox = QtWidgets.QComboBox(Dialog)
        self.comboBox.setGeometry(QtCore.QRect(110, 50, 151, 41))
        self.comboBox.setObjectName("comboBox")
        self.comboBox.addItem("")
        self.comboBox.addItem("")
        self.comboBox.addItem("")
        self.comboBox.addItem("")
        self.label_2 = QtWidgets.QLabel(Dialog)
        self.label_2.setGeometry(QtCore.QRect(20, 50, 81, 41))
        self.label_2.setObjectName("label_2")

        self.retranslateUi(Dialog)
        QtCore.QMetaObject.connectSlotsByName(Dialog)

    def retranslateUi(self, Dialog):
        _translate = QtCore.QCoreApplication.translate
        Dialog.setWindowTitle(_translate("Dialog", "Dialog"))
        self.pushButton.setText(_translate("Dialog", "采集"))
        self.label.setText(_translate("Dialog", "关键字"))
        self.comboBox.setItemText(0, _translate("Dialog", "前5页"))
        self.comboBox.setItemText(1, _translate("Dialog", "前10页"))
```

```
        self.comboBox.setItemText(2, _translate("Dialog", " 前 15 页 "))
        self.comboBox.setItemText(3, _translate("Dialog", " 前 20 页 "))
        self.label_2.setText(_translate("Dialog", " 采集页数 "))
```

14.5 MVC——视图

软件开发应遵从 MVC 结构设计，下面简单介绍一下 MVC 结构。

MVC 全名是 Model View Controller，是模型（Model）—视图（View）—控制器（Controller）的缩写，是一种软件设计典范，用一种业务逻辑、数据、界面显示分离的方法组织代码，将业务逻辑聚集到一个部件里面，在改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑。MVC 被独特的发展起来，用于在一个逻辑的图形化用户界面的结构中映射传统的输入、处理和输出功能。

在项目中，我们已经得到 taobao_v.py 文件，在 taobao_v.py 文件所在目录下创建 taobao_vc.py 文件，该文件主要是将 taobao_v.py 界面以 Python 程序运行方式呈现出来。taobao_vc.py 代码如下：

```
from PyQt5 import QtCore, QtGui, QtWidgets
# 导入 taobao.py 的 Ui_Dialog
from taobao import Ui_Dialog
import sys
# 继承 taobao.py 的 Ui_Dialog
class taobao_control(QtWidgets.QMainWindow, Ui_Dialog):
    def __init__(self, parent=None):
        super(taobao_control, self).__init__(parent)
        self.setupUi(self)

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    taobao_control = taobao_control()
    taobao_control.show()
    sys.exit(app.exec_())
```

taobao_vc.py 继承了 taobao_v.py 里的 Ui_Dialog，两个文件共同实现一个软件的界面开发。因为 taobao_v.py 使用 Qt Designer 设计界面并通过 PyCharm 外部工具

PyUIC 将其转化成 py 源文件。转换后的 py 文件只是一个静态软件界面，因此还需要编写对象间的通信机制（信号 / 槽），创建 taobao_vc.py 的目的是编写对象间的信号 / 槽，将信号 / 槽和软件设计在不同文件中实现。其主要原因是，每次更新界面设计时，更新的代码会覆盖原有的界面代码，如果将信号 / 槽与软件界面代码写到同一个文件，每次更新软件设计，开发者都需要重新编写对应的信号 / 槽，这样对维护和修改极其不便。因此，将两者在不同文件中实现可以将界面设计和逻辑开发分离。

在 taobao_vc.py 中，对采集按钮添加响应事件，代码如下：

```
from PyQt5 import QtCore, QtGui, QtWidgets
# 导入 taobao.py 的 Ui_Dialog
from taobao_v import Ui_Dialog
# 导入逻辑功能，MVC 里面的 C
from taobao_c import get_info
import sys
# 继承 taobao.py 的 Ui_Dialog
class taobao_control(QtWidgets.QMainWindow, Ui_Dialog):
    def __init__(self, parent=None):
        super(taobao_control, self).__init__(parent)
        self.setupUi(self)
        # 添加响应事件
        self.pushButton.clicked.connect(self.collect_data)

    def collect_data(self):
        # 获取多个关键字，返回关键字列表
        get_keyword_list = self.plainTextEdit.toPlainText().
split('\n')
        # 获取采集的页数
        get_page = (self.comboBox.currentIndex()+1)*5
        # 每一页的数据要请求 44 次
        get_page = get_page*44
        get_info(get_keyword_list, get_page, [])
        print('Done')

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    taobao_control = taobao_control()
```

```
    taobao_control.show()
    sys.exit(app.exec_())
```

上述代码中添加了按钮的响应事件，单击按钮就会触发 collect_data() 函数，该函数先获取并处理用户在软件界面上输入的数据，然后调用 get_info() 函数，爬取淘宝的商品信息，get_info() 函数是 MVC 里控制器的函数，具体内容会在 14.6 节讲解。

从整个视图功能分析，视图由 taobao_v.py 和 taobao_vc.py 共同实现。前者是由 Qt Designer 设计界面并通过 PyCharm 外部工具 PyUIC 将其转化成 py 源文件，主要承担了软件的控件布局和命名；后者是继承前者，主要在前者的基础上编写通信机制（信号 / 槽）。这开发设计可使界面设计和逻辑开发分离，便于软件修改和维护。

14.6 MVC——控制器

回顾第 12 章可以知道，整个项目代码都是在一个 py 文件中完成的。在此，我们将代码功能拆分，使其符合 MVC 结构。控制器是整个 MVC 的枢纽部分，承担了视图和模型衔接和通信的功能。在 14.5 节，代码调用了 get_info() 函数，该函数用于实现控制器和视图的衔接和通信。

将控制器代码文件命名为 taobao_c.py，代码如下：

```
import requests
import json
import csv
from taobao_m import get_auctions_info

def get_info(get_keyword_list, get_page, auctions_distinct):
    for k in get_keyword_list:
        # 新建 csv 文件
        file_name = k + ".csv"
        with open(file_name, "w", newline='') as csvfile:
            writer = csv.writer(csvfile)
            # 写入数据
            writer.writerow(['标题', '价格', '销量', '店铺', '区域'])
            csvfile.close()
```

```
            for p in range(get_page):
                url = 'https://s.taobao.com/api?callback=jsonp804&m=
                       customized&q=%s&s=%s' %(k, p)
                r = requests.get(url)
                response = r.text
                response = response.split('(')[1].split(')')[0]
                response_dict = json.loads(response)
                response_auctions = response_dict['API.
CustomizedApi']['itemlist']['auctions']
                # 数据存储
                auctions_distinct = get_auctions_info(response_
auctions,file_name,auctions_distinct)
```

函数 get_info() 主要用于创建 CVS 文件、数据抓取和调用函数 get_auctions_info() 对数据入库。调用函数 get_auctions_info() 是衔接模型部分，用于实现控制器和模型之间的通信。

14.7 MVC——模型

在 14.6 节看到，控制器的代码调用函数 get_auctions_info()，该函数主要对传递的参数实现数据存储。将数据存储（模型）代码文件命名为 taobao_m.py，代码如下：

```
import csv
def get_auctions_info(response_auctions_info, file_name,
    auctions_distinct):
    with open(file_name, "a", newline='') as csvfile:
        # 生成 csv 对象，用于写入 csv 文件
        writer = csv.writer(csvfile)
        for i in response_auctions_info:
            # 判断数据是否已经记录
            if str(i['raw_title']) not in auctions_distinct:
                # 写入数据
                writer.writerow([i['raw_title'], i['view_price'],
                                 i['view_sales'], i['nick'],
                                 i['item_loc']])
```

```
                auctions_distinct.append(str(i['raw_title']))
        csvfile.close()
    return auctions_distinct
```

从代码中可以看到，函数 get_auctions_info() 主要是将参数 response_auctions_info 写入 CSV 文件，并将每次写入的信息记录在参数 auctions_distinct 中，最后返回参数 auctions_distinct 给控制器。控制器再次调用 get_auctions_info() 的时候，将上一次返回的 auctions_distinct 内容作为当次的函数参数。变量 auctions_distinct 在控制器和模型之间不停地交互通信。

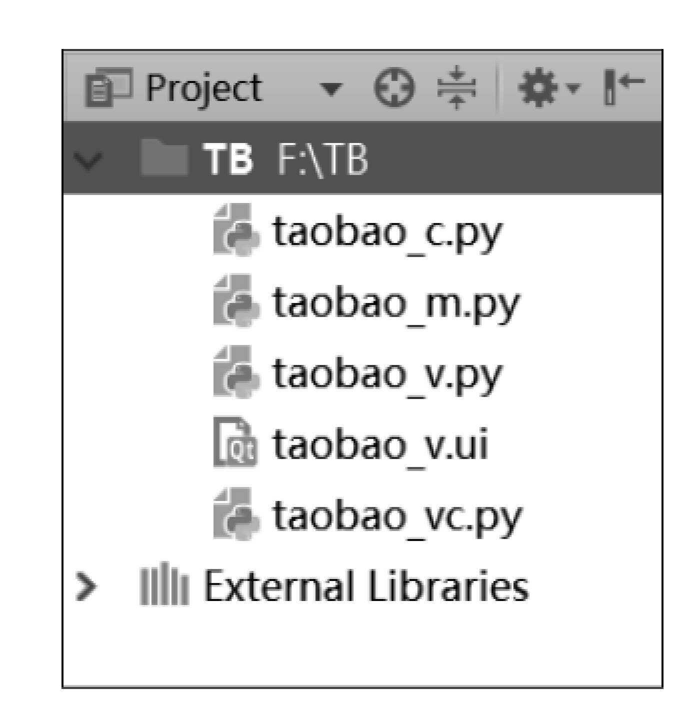


图 14-9 项目目录结构

最终整个项目的目录如图 14-9 所示。

整个目录下共有 5 个文件，分别介绍如下。

- taobao_c.py：MVC 里面的控制器部分，主要衔接视图和模型。
- taobao_m.py：MVC 里面的模型部分，主要实现数据存储。
- taobao_v.py 和 taobao_vc.py：共同组成 MVC 的视图部分，后者继承前者，并由后者负责项目的运行和启动。
- taobao_v.ui：Qt Designer 界面设计文件，可转换成 taobao_v.py 文件。

14.8 扩展思路

到目前为止，项目已实现 MVC 结构化开发。从用户使用的角度来看，软件功能存在以下两个问题：

（1）软件界面过于简单，用户体验不强。

（2）数据爬取速度过慢，关键字过多或页数过多会导致软件崩溃。

解决思路：

（1）在软件界面增加进度条功能，根据爬取速度制定进度反馈。

（2）在界面中增加状态提示功能，比如网络中断提示、关键字为空提示等。

（3）在逻辑上增加分布式功能，提高爬虫效率。

（4）软件界面美化，设置背景图、背景颜色、按钮图标等。

本书到这里只提供扩展和优化思路，不做过多讲解，希望读者能在此基础上结合整章内容对本项目进行扩展和优化。

14.9 本章小结

PyQt5 是一套绑定 Qt5 的应用程序框架，由 Python 语言实现，已经有超过 620 个类和 6000 个函数与方法。PyQt5 是一个运行在所有主流操作系统上的多平台组件，包括 UNIX、Windows 和 Mac OS。PyQt5 是双重许可的，开发者可以选择 GPL 和商业许可。

通过本章的学习，读者应该掌握以下内容：

1. PyQt5 的安装及其环境搭建

（1）安装 PyQt5：pip install PyQt5。

（2）安装 Qt Designer：pip install PyQt5-tools。

（3）配置 PyChram 环境。

2. Qt Designer 模板说明

（1）MainWindow 是主界面，一个窗口是父 / 子 hiearchy 的顶部，通常显示标题栏和边框。底层窗口系统（Windows、KDE、GNOME 等）将为窗口提供策略，如标题栏 / 边框样式、布局和焦点等。

（2）Widget 是小部件，屏幕上的一个矩形区域，用于显示和用户交互，包括按钮、滑块、视图、对话框和窗口等。所有窗口小部件将在屏幕上显示某些内容，许多窗口小部件也将接收来自键盘或鼠标输入的内容。“Widget”一词来自 UNIX，在 Windows 上称为“控件”。

（3）Dialog 对话框通常是临时的，可以设置不同的标题栏外观，主要用于通知或收集输入窗口，并且通常底部或右侧有 OK、Cancel 等按钮。

3. MVC 的概念

MVC 全名是 Model View Controller，是模型（Model）－视图（View）－控制器（Controller）的缩写，一种软件设计典范，用业务逻辑、数据、界面显示分离的方法组织代码，将业务逻辑聚集到一个部件里面，在改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑。MVC 被独特地发展起来，用于在一个逻辑的图形化用户界面结构中映射传统的输入、处理和输出功能。

4. 项目目录文件

- taobao_c.py：MVC 里面的控制器部分，主要衔接视图和模型。
- taobao_m.py：MVC 里面的模型部分，主要实现数据存储。
- taobao_v.py 和 taobao_vc.py：共同组成 MVC 的视图部分，后者继承前者，并由后者负责项目的运行和启动。
- taobao_v.ui：Qt Designer 界面设计文件，可转换成 taobao_v.py 文件。

第 15 章

项目实战：12306 抢票

15.1 分析说明

12306 抢票是爬虫开发中非常经典的一个项目。官方为了打击黄牛囤票，网站不断地更新升级，各类抢票软件也不断地修正更改，两者周而复始，印证了一句话“程序员都是在互相伤害”。

这种抢票类爬虫的开发思路与用户在浏览器的购票操作一致，只不过是编写代码来完成购票流程，可以理解为通过程序来模拟用户在浏览器上购买车票。

在本项目中，按照购买火车票的流程：用户登录→查询车票信息→选择班次→填写乘车人员信息→提交并生成订单，制定爬虫功能开发顺序。

（1）验证码验证。

（2）用户登录与验证。

（3）查询车票。

（4）预订车票。

（5）提交订单。

（6）生成订单。

15.2 验证码验证

在 12306 购买火车票的时候，首先需要用户登录，除了需要输入账号、密码之外，还设有图片验证码，验证码的验证方式是根据图片中的问题选择正确的答案，当验证码和账号信息正确时，才能登录成功。

在爬虫中实现登录功能，首先分析网站的登录事件所触发的请求信息。在 Chrome 浏览器中访问 12306 的用户登录界面（https://kyfw.12306.cn/otn/login/init）。该登录界面除了账号、密码输入框之外，还有一个图片验证码，图片验证码是由一个问题描述和 8 组图片组成的。打开开发者工具，在登录界面输入账号、密码并选择错误的验证码答案，最后单击登录按钮，可以看到有两个请求信息，如图 15-1 所示。

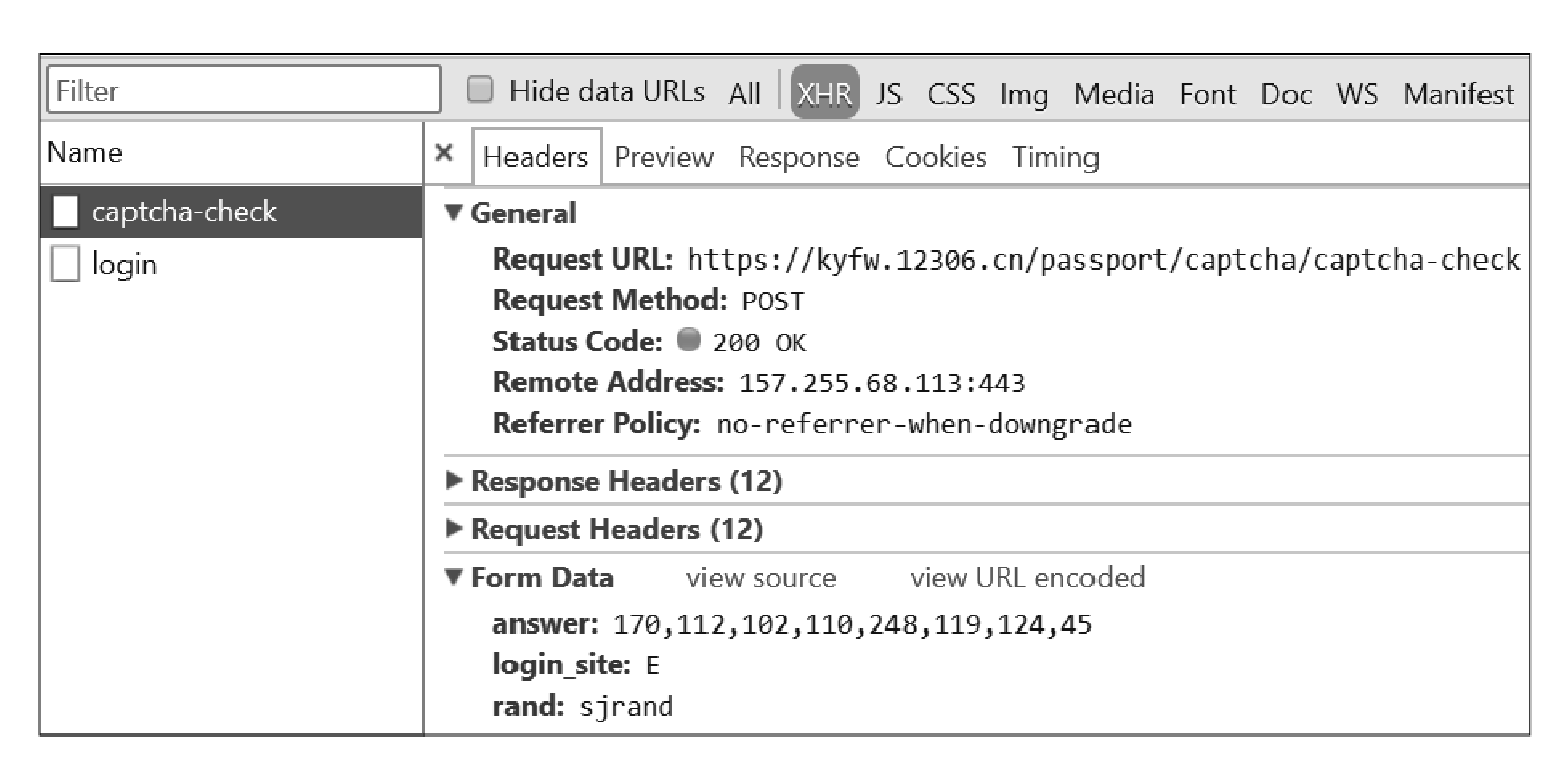


图 15-1　验证码验证

从图 15-1 可以看到，当输入正确的账号、密码并选择错误的验证码答案登录后，会触发两个 POST 请求，其中第一个请求链接是 https://kyfw.12306.cn/passport/captcha/

captcha-check，从 URL 组成和响应内容分析，该请求信息的作用是验证用户选择的答案与验证码答案是否符合，如图 15-2 所示。

```
×  Headers  Preview  Response  Cookies  Timing
▼{result_message: "验证码校验失败", result_code: "5"}
   result_code: "5"
   result_message: "验证码校验失败"
```

图 15-2 验证码校验结果

从图 15-1 可知，验证码校验请求有三个参数，分别是 answer、login_site 和 rand。单从一次请求信息是无法找出请求参数的变化规律的，不妨重复多次上述操作，观察请求参数的变化来判断数据规律。通过多次操作（输入正确的账号、密码，并选择错误而不重复的验证码答案），发现参数 login_site 和 rand 的参数值是固定不变的，而参数 answer 会根据每次选择的答案不同而不断地变化。

为了进一步找出参数 answer 的变化规律，尝试以下方法：

（1）第一次只选择第一组图片，参数 answer 的值为 40,40。

（2）第二次只选择第二组图片，参数 answer 的值为 114,35。

（3）第三次只选择第三组图片，参数 answer 的值为 192,39。

（4）以此类推，第四、第五、第六、第七和第八组图片分别对应 257,36、42,115、119,107、185,124 和 272,117。也就是说，每组图片对应一组数字。

通过多次试验发现，同一组图片，根据单击位置的不同，参数 answer 的值随之变化，如第一组图片，单击位置分别在左上方和左下方，其参数 answer 的值有所不同。根据这样的变化，每组数字应该代表一个坐标位置，每组图片代表一定的区域范围，只要坐标位置在图片的区域范围内，这组数据就代表这张图片。这种验证码称之为坐标验证码，这种坐标系的验证码属于图片验证码里面的一种类型，对于这种验证码目前还没有很好的解决方案，只能通过人为输入正确的坐标位置来完成验证。

综合分析，可以确定验证码里面的 8 组图片的坐标位置（每组图片的坐标位置不是唯一的，只要坐标位置在图片的区域内即可），验证码校验代码如下：

```
import requests
# 坐标参考：40,40,114,35,192,39,257,36,42,115,119,107,185,124,272,117
code_list={
    '1': '40,40,',
    '2': '114,35,',
    '3': '192,39,',
    '4': '257,36,',
    '5': '42,115,',
    '6': '119,107,',
    '7': '185,124,',
    '8': '272,117'
}
# 创建会话
session = requests.session()
# 请求头
headers = { 'User-Agent':
            'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 '
            '(KHTML, like Gecko) Chrome/63.0.3218.0 Safari/537.36',
            'Referer':
            'https://kyfw.12306.cn/otn/login/init'}
# 验证码图片的 URL
url = 'https://kyfw.12306.cn/passport/captcha/captcha-image?
login_site=E&module=login&rand=sjrand'
# 忽略证书验证
r = session.get(url, headers=headers, verify=False)
# 下载验证码图片
F = open('code.png','wb')
f.write(r.content)
f.close()
# 输入验证码图片位置，每组图片用英文逗号隔开
code = input("请输入验证码：")
get_code = ''
for i in code.split(','):
    # 根据输入每组图片的组号获取对应的坐标位置
    get_code += code_list[i]
# 验证码校验
data = {
```

```
        'answer': get_code,
        'login_site': 'E',
        'rand': 'sjrand'
    }
    url = 'https://kyfw.12306.cn/passport/captcha/captcha-check'
    r = session.post(url, data=data)
    print(r.text)
```

整段代码实现了两次请求，第一次请求是下载验证码图片，第二次请求是对验证码进行校验。代码细节如下。

- code_list：使用字典数据格式，主要用于验证码识别，用户可直接输入每组图片的组号获取对应的坐标位置。
- session = requests.session()：创建一个持久化会话对象，确保每一次请求在同一个会话中。
- verify=False：忽略证书验证。如果没有安装 12306 网站的根证书，爬取过程中会提示连接不安全而导致无法访问，一般忽略证书验证即可解决。
- 验证码识别：根据验证码的问题找出图片所在组号的位置即可。图片位置顺序是从上到下再从左到右，组号从 1 到 8 依次排序。如果有多个组号，组号之间就用英文的逗号隔开。

验证码验证：将输入的字符串以逗号分割后，根据图片位置找到对应的图片坐标，最后将坐标拼接起来就得到参数 answer。

运行上述代码，如图 15-3 所示。

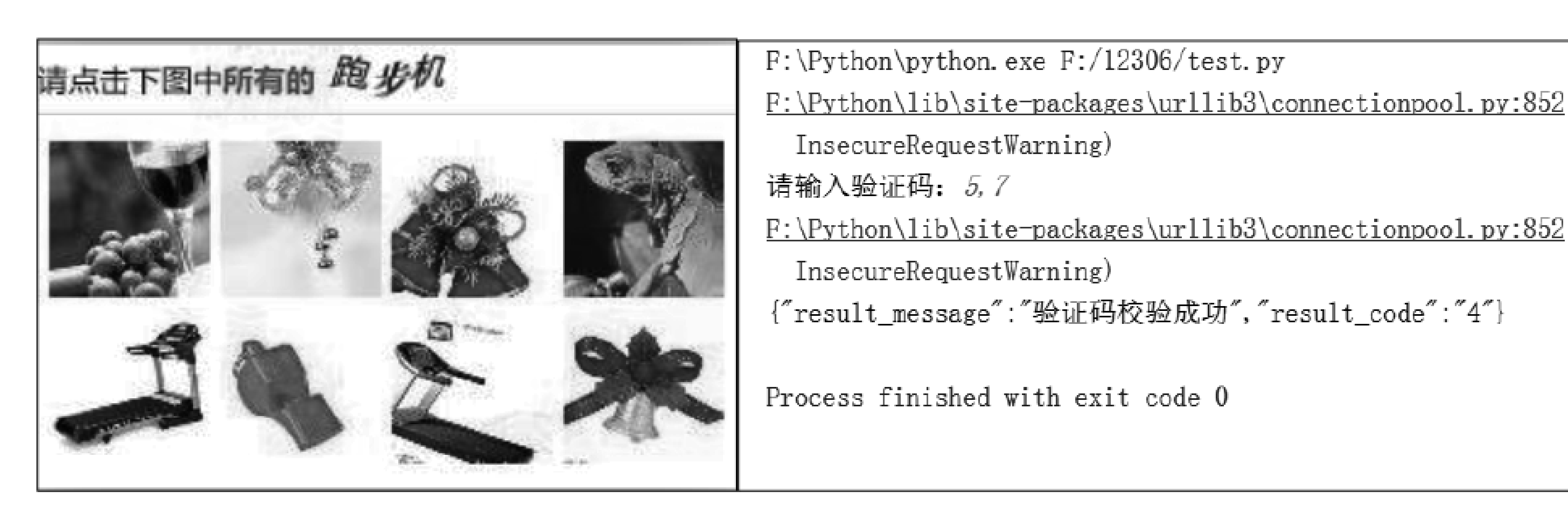


图 15-3 验证结果

15.3 用户登录与验证

完成验证码验证，下一步是实现用户登录功能。从图 15-1 可知，单击登录按钮会触发两个 POST 请求，其中第二个是用户登录请求，对该请求进行分析，如图 15-4 所示。

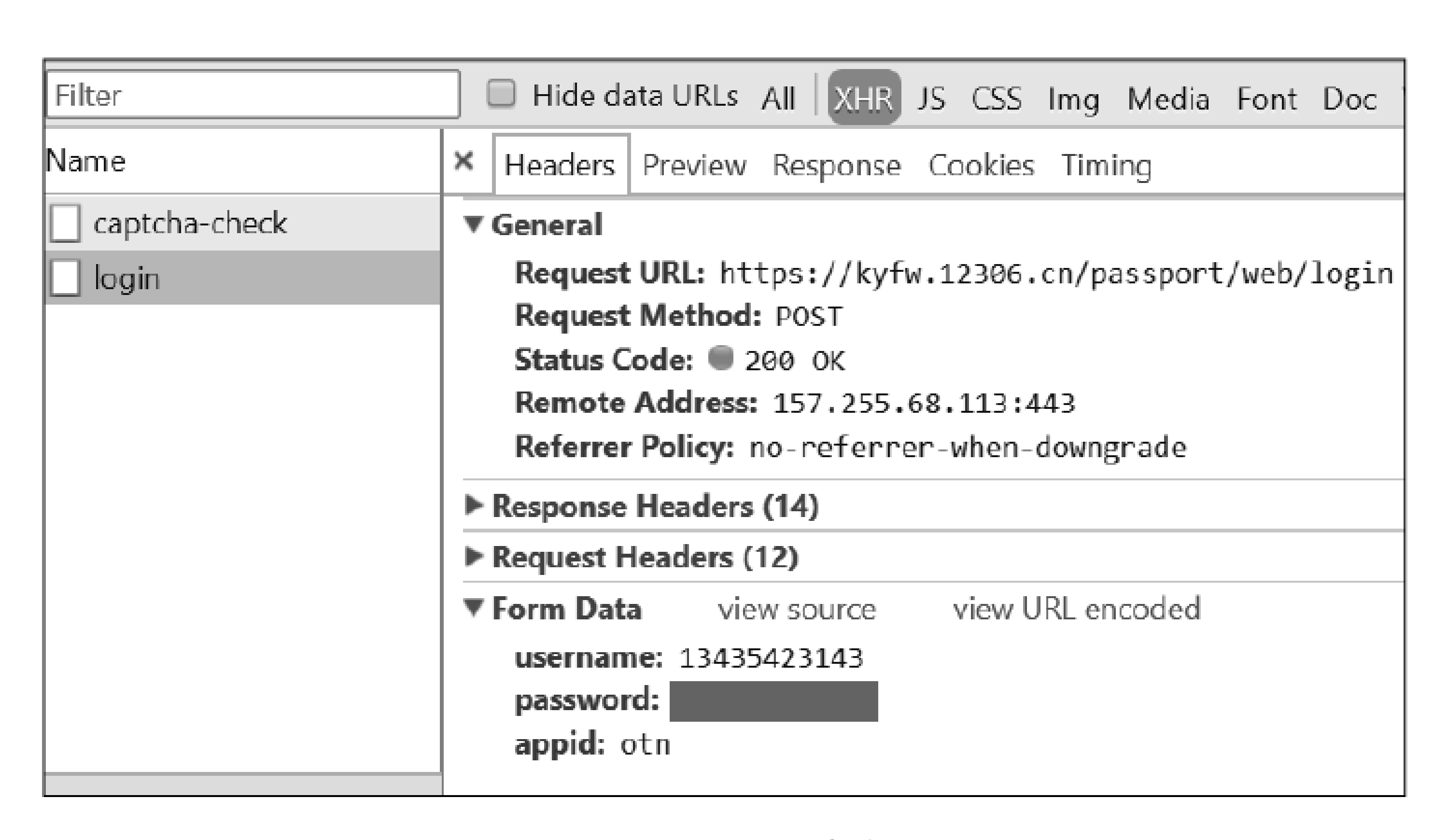


图 15-4 登录请求

登录请求的参数有 username、password 和 appid，而且 username 和 password 没有经过加密处理，参数 appid 是固定不变的。那么，用户登录的代码如下：

```
url ='https://kyfw.12306.cn/passport/web/login'
data = {
    'username': '13435423143',
    'password': 'XXXXXX',
    'appid': 'otn'
}
r = session.post(url, data=data)
print(r.text)
```

由于用户登录的代码无法单独运行，因此我们将验证码验证和用户代码整合在一起，只要将用户登录的代码添加到验证码的代码下面即可，运行结果如图 15-5 所示。

完成用户登录后，接着回到登录界面，当输入正确的账号、密码和验证码之后，单击登录按钮，触发两个请求之后，发现网页会自动跳转，在网页跳转时，在开发者工具捕捉到两个新的POST 请求，但这两个请求只在一瞬间出现，等网页跳转完成之后，请求信息就会被清理掉。

```
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request
  InsecureRequestWarning)
请输入验证码：5,6
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request
  InsecureRequestWarning)
{"result_message":"验证码校验成功","result_code":"4"}
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request
  InsecureRequestWarning)
{"result_message":"登录成功","result_code":0,"uamtk":"0njyEiC1OmhGTRb5W3vPv9E3n45XPhAk-5Msuw4mEnwsd1110"}
```

图 15-5 用户登录信息

这是因为 HTTP 的 302 跳转，网页跳转之后，Chrome 浏览器将之前捕捉到的请求清空，然后重新捕捉新页面的请求。遇到这种情况的时候，只能在网页跳转的期间内单击开发者工具，然后按 Ctrl+E 停止 Network 对请求的捕捉，这样就能保留之前的请求信息。除此之外，读者还可以使用 Fiddler 对网站抓包。

回到本项目中，在网页跳转之前，Chrome 浏览器捕捉的请求信息如图 15-6 和图 15-7 所示。

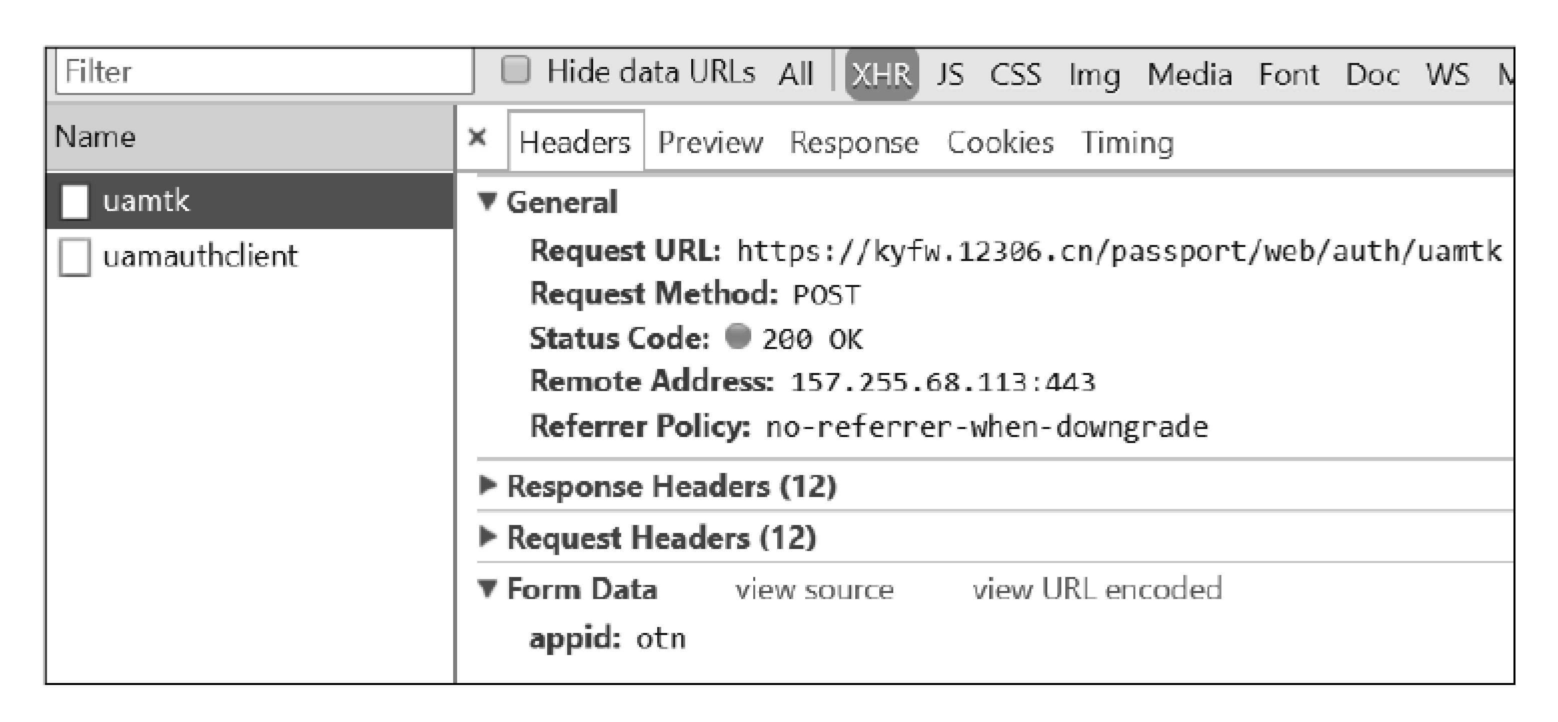


图 15-6 用户登录验证一

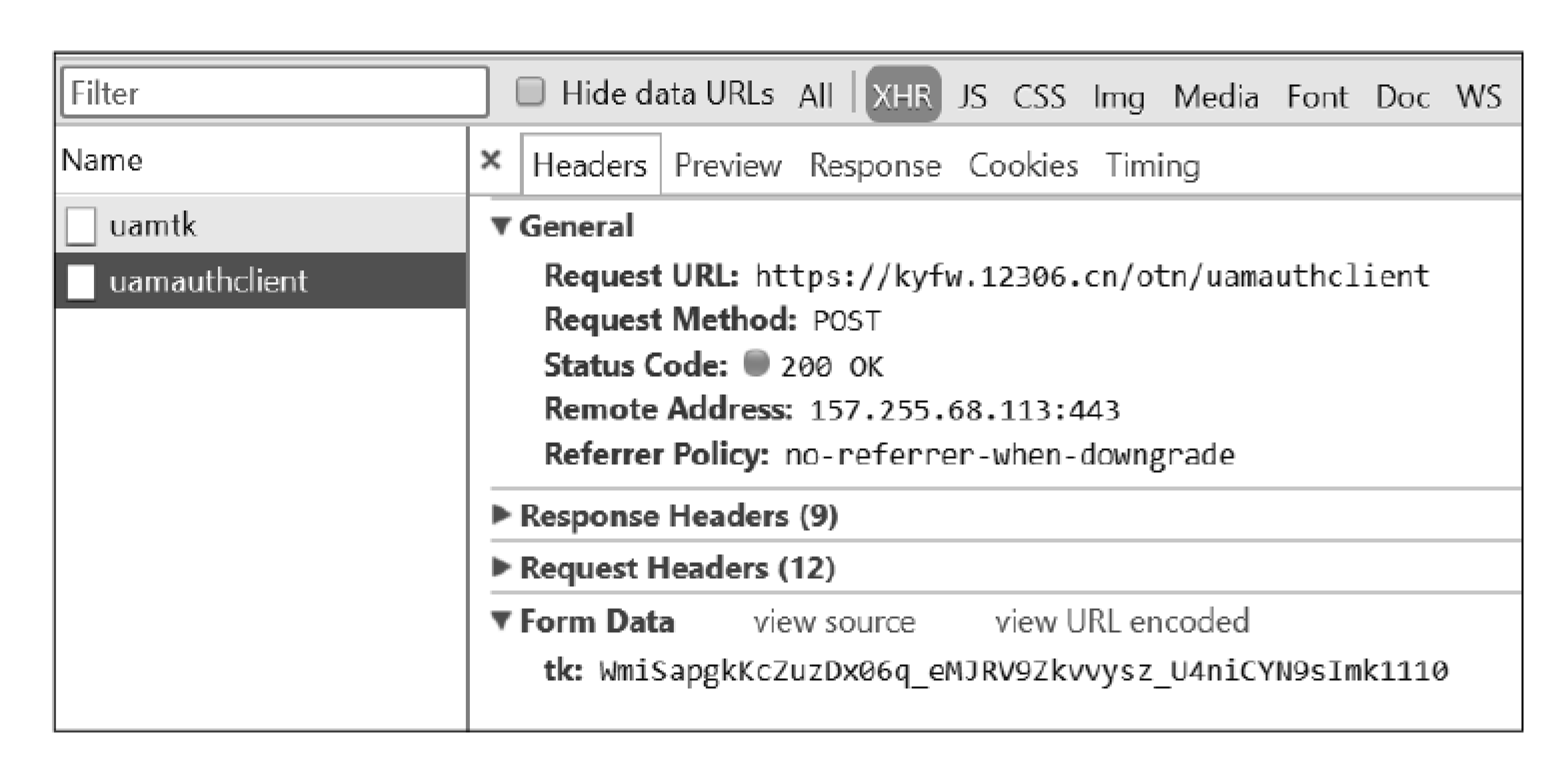


图 15-7 用户登录验证二

从图 15-6 和图 15-7 可知，网页跳转时，发生了两次 POST 请求，而且两者只有一个请求参数，图 15-6 的请求参数是固定值，而图 15-7 的请求参数是变化值。

现在无法确定图 15-7 请求参数的由来。一般来说，请求参数主要的来源如下：

（1）Doc 标签的 HTML 内容，可以复制参数值的内容，然后在 HTML 中快速查找参数是否存在。

（2）XHR 标签的请求信息，参数可能由其他的请求信息生成。

（3）JS 标签的请求信息，需要对 JavaScript 代码进行解读，参数有可能由 JavaScript 生成。

（4）特殊数据，如随机数和时间戳。随机数大多数都是以小数为主的，大多数随机数可以视为固定不变的参数；时间戳通常以 150XXXXX 开头，长度一般为 9 ～ 16 位不等。

根据上述查找方法结合实际分析，当前只有两个 POST 请求，第二个请求信息的请求参数很可能来自于第一个请求信息的响应内容。

在上述已完成的代码中添加以下代码，实现图 15-6 的请求：

```
url = 'https://kyfw.12306.cn/passport/web/auth/uamtk'
data = {
    'appid': 'otn'
}
r = session.post(url, data=data)
print(r.text)
```

运行代码，结果如图 15-8 所示。

```
  InsecureRequestWarning)
请输入验证码：4,5
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request is being made.
  InsecureRequestWarning)
{"result_message":"验证码校验成功","result_code":"4"}
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request is being made.
  InsecureRequestWarning)
{"result_message":"登录成功","result_code":0,"uamtk":"9SHmBE6TX_imlaKVaUD_TwqxT3WLyRDR-r05nMwPiPw1p1110"}
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request is being made.
  InsecureRequestWarning)
{"result_message":"验证通过","result_code":0,"apptk":null,"newapptk":"tg_aFvSVVZAtN8srMpIXMM4gHIRb-BGL-Euop4hHVcket1110"}
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request is being made.
```

图 15-8　用户登录验证结果一

从最后的输出结果分析，newapptk 的数据格式和图 15-7 的请求参数比较符合。尝试使用 newapptk 的数据作为图 15-7 的请求参数，在上述代码中添加以下代码：

```
# newapptk 是图 15-6 请求之后的响应结果
newapptk = r.json()['newapptk']
url = 'https://kyfw.12306.cn/otn/uamauthclient'
data = {
    'tk': newapptk
}
r = session.post(url, data=data)
print(r.text)
```

运行结果如图 15-9 所示。

```
请输入验证码：4,5
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request is being made. Adding
  InsecureRequestWarning)
{"result_message":"验证码校验成功","result_code":"4"}
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request is being made. Adding
  InsecureRequestWarning)
{"result_message":"登录成功","result_code":0,"uamtk":"9SHmBE6TX_imlaKVaUD_TwqxT3WLyRDR-r05nMwPiPwlpl110"}
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request is being made. Adding
  InsecureRequestWarning)
{"result_message":"验证通过","result_code":0,"apptk":null,"newapptk":"tg_aFvSVVZAtN8srMpIXMM4gHIRb-BGL-Euop4hHVcket1110"}
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request is being made. Adding
  InsecureRequestWarning)
{"apptk":"tg_aFvSVVZAtN8srMpIXMM4gHIRb-BGL-Euop4hHVcket1110","result_code":0,"result_message":"验证通过","username":"黄永祥"}
```

图 15-9 用户登录验证结果二

根据图 15-9 的运行结果得知，第二次 POST 的请求参数（见图 15-7）来自于第一次 POST 请求（见图 15-6）的响应内容，也就说这两个请求是紧密联系的，共同完成用户登录验证功能。

用户登录网站由三部分功能组成：验证码验证→用户登录→用户验证。对这三部分功能进行优化和整理，代码如下：

```
import requests
def login(username, password):
    # 坐标参考：40,40,114,35,192,39,257,36,42,115,119,107,185,124,
272,117
    code_list = {
        '1': '40,40,',
        '2': '114,35,',
```

```
        '3': '192,39,',
        '4': '257,36,',
        '5': '42,115,',
        '6': '119,107,',
        '7': '185,124,',
        '8': '272,117'
    }
    # 请求头
    headers = { 'User-Agent':
                'Mozilla/5.0 (Windows NT 10.0; WOW64)
                    AppleWebKit/537.36 '
                '(KHTML, like Gecko) Chrome/63.0.3218.0
                    Safari/537.36',
                'Referer':
                'https://kyfw.12306.cn/otn/login/init'}

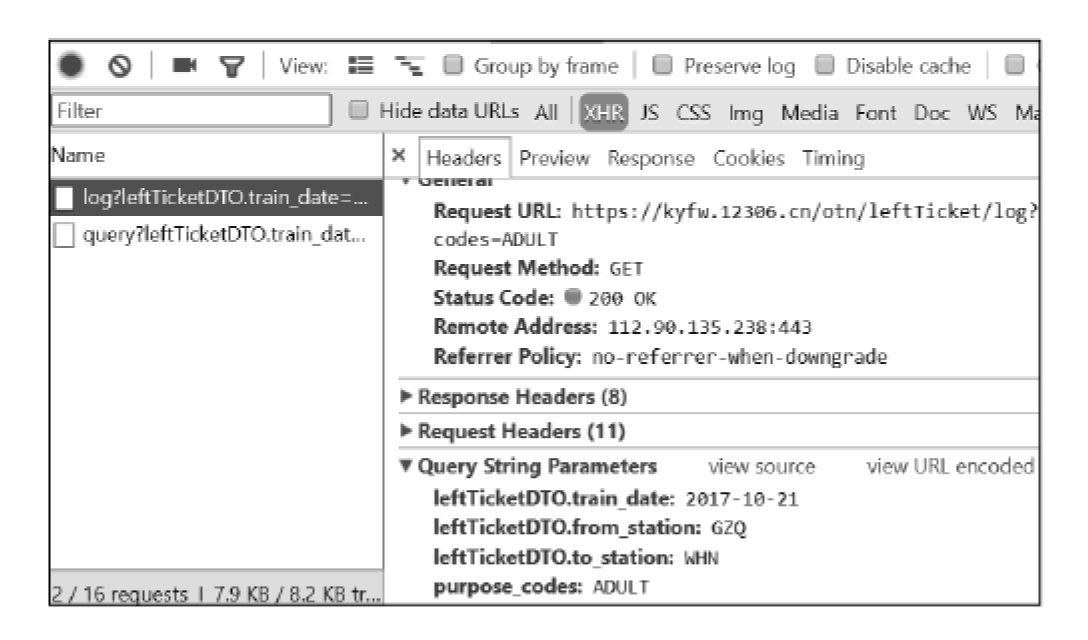
    url = 'https://kyfw.12306.cn/passport/captcha/captcha-image?
login_site=E&module=login&rand=sjrand'
    # 忽略证书验证
    r = session.get(url, headers=headers, verify=False)
    # 下载验证码图片
    f = open('code.png','wb')
    f.write(r.content)
    f.close()
    # 输入验证码图片位置，多个验证码用英文逗号分开
    code=input(" 请输入验证码：")
    get_code = ''
    for i in code.split(','):
        # 根据输入每组图片的组号获取对应的坐标位置
        get_code += code_list[i]
    # 验证码校验
    data={
        'answer':get_code,
        'login_site':'E',
        'rand':'sjrand'
    }
    url = 'https://kyfw.12306.cn/passport/captcha/captcha-check'
```

```
        r = session.post(url, data=data)
        print(r.text)
        if '验证码校验失败' not in str(r.text):
            # 用户登录
            url = 'https://kyfw.12306.cn/passport/web/login'
            data = {
                'username': username,
                'password': password,
                'appid': 'otn'
            }
            r = session.post(url, data=data)
            print(r.text)
            if '密码输入错误' not in str(r.text):
                # 登录验证第一次请求
                url = 'https://kyfw.12306.cn/passport/web/auth/uamtk'
                data = {
                    'appid': 'otn'
                }
                r = session.post(url, data=data)
                # 登录验证第二次请求
                newapptk = r.json()['newapptk']
                url = 'https://kyfw.12306.cn/otn/uamauthclient'
                data = {
                    'tk': newapptk
                }
                r=session.post(url, data=data)
                print(r.text)
      return True
            else:
                return False
        return False
    if __name__=='__main__':
        # 创建持久化会话对象
        session = requests.session()
        username = 'xxxxxxx'
        password = 'xxxxxxx'
        login_info = login(username, password)
        print(session)
```

15.4 查询车次

查询车次信息首先要输入出发地、目的地和出发日期，完成信息输入后，单击“查询”按钮，网站根据输入的信息返回相应的车次信息。

从一个正常的车次查询流程中发现，实际上网站与用户的交互是在用户单击“查询”按钮时发生的。在开发者工具捕捉到的请求如图 15-10 和图 15-11 所示。



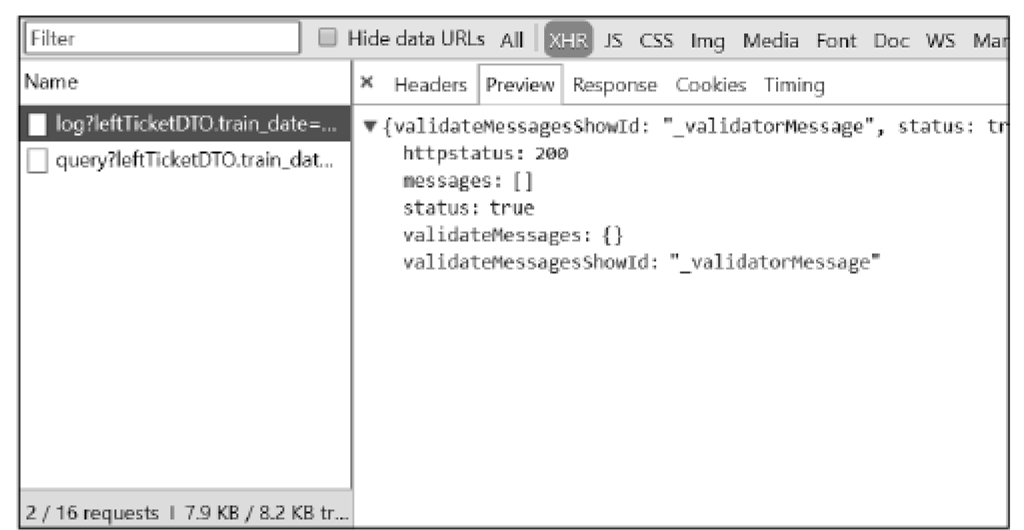
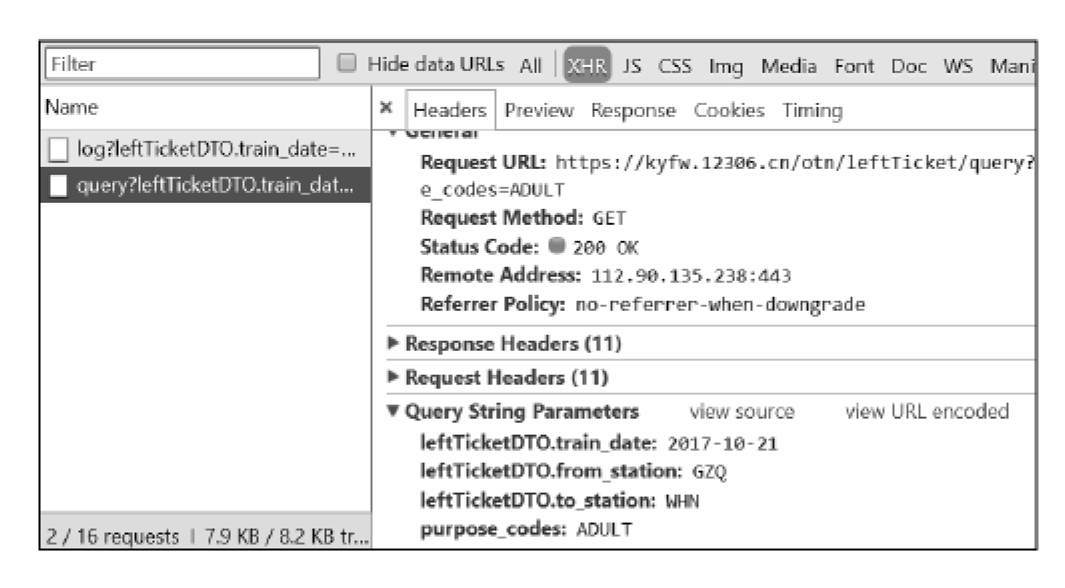


图 15-10　车票查询请求及响应内容一



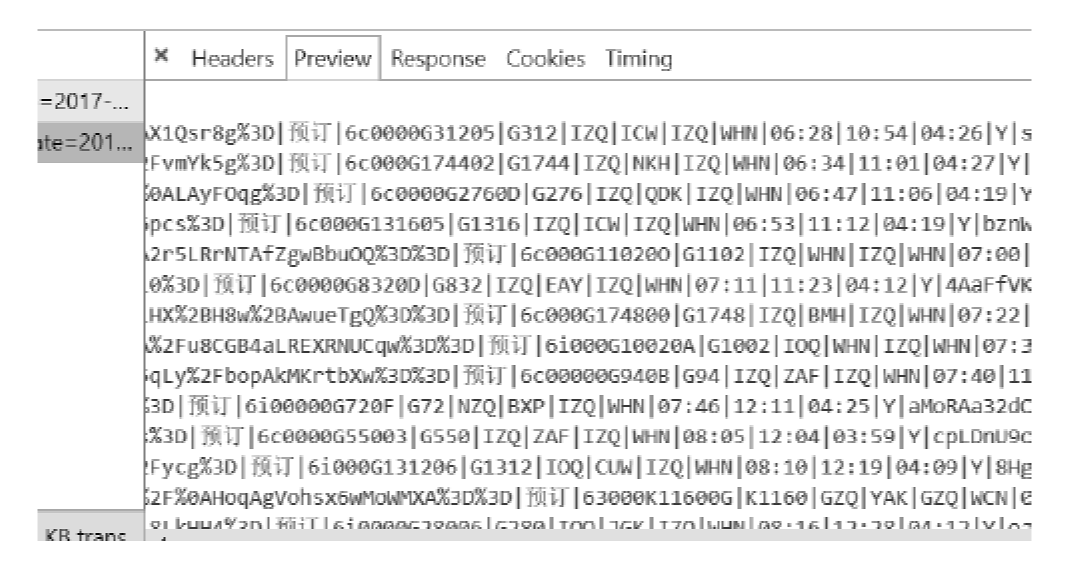


图 15-11　车票查询请求及响应内容二

对请求信息分析可知，用户在单击“查询”按钮后会发送两个 GET 请求，两个 GET 请求的参数是一致的。再查看两者的返回数据，图 15-10 的返回数据并没有太大用处；图 15-11 的返回数据内容较多，而且与网页显示的车次信息（见图 15-12）对比发现，两者的数据是可以相互匹配的。也就是说，网页上的车次信息由图 15-11 的请求信息生成并按照某种方式渲染到网页上。

在代码中实现车次查询，首先要找到请求参数的数据来源。从图 15-10 和图 15-11 的参数可知，出发地和目的地都是英文字母，前两个字母是由城市名的拼音首字母组成的，最后一个字母无法确认。

G312	广州南 武汉	06:28 10:54	04:26 当日到达	8	有	有	--
G1744	广州南 武汉	06:34 11:01	04:27 当日到达	5	1	有	--
G276	广州南 武汉	06:47 11:06	04:19 当日到达	20	有	有	--
G1316	广州南 武汉	06:53 11:12	04:19 当日到达	4	无	有	--
G1102	广州南 武汉	07:00 11:18	04:18 当日到达	5	14	有	--
G832	广州南 武汉	07:11 11:23	04:12 当日到达	18	有	有	--

图 15-12 查询车次信息

根据 15.3 节的请求参数查找方法，分别在 Doc、XHR、JS 标签查找和分析各个请求信息。我们在 JS 标签某个请求的响应内容中找到城市的字母编号，如图 15-13 所示。

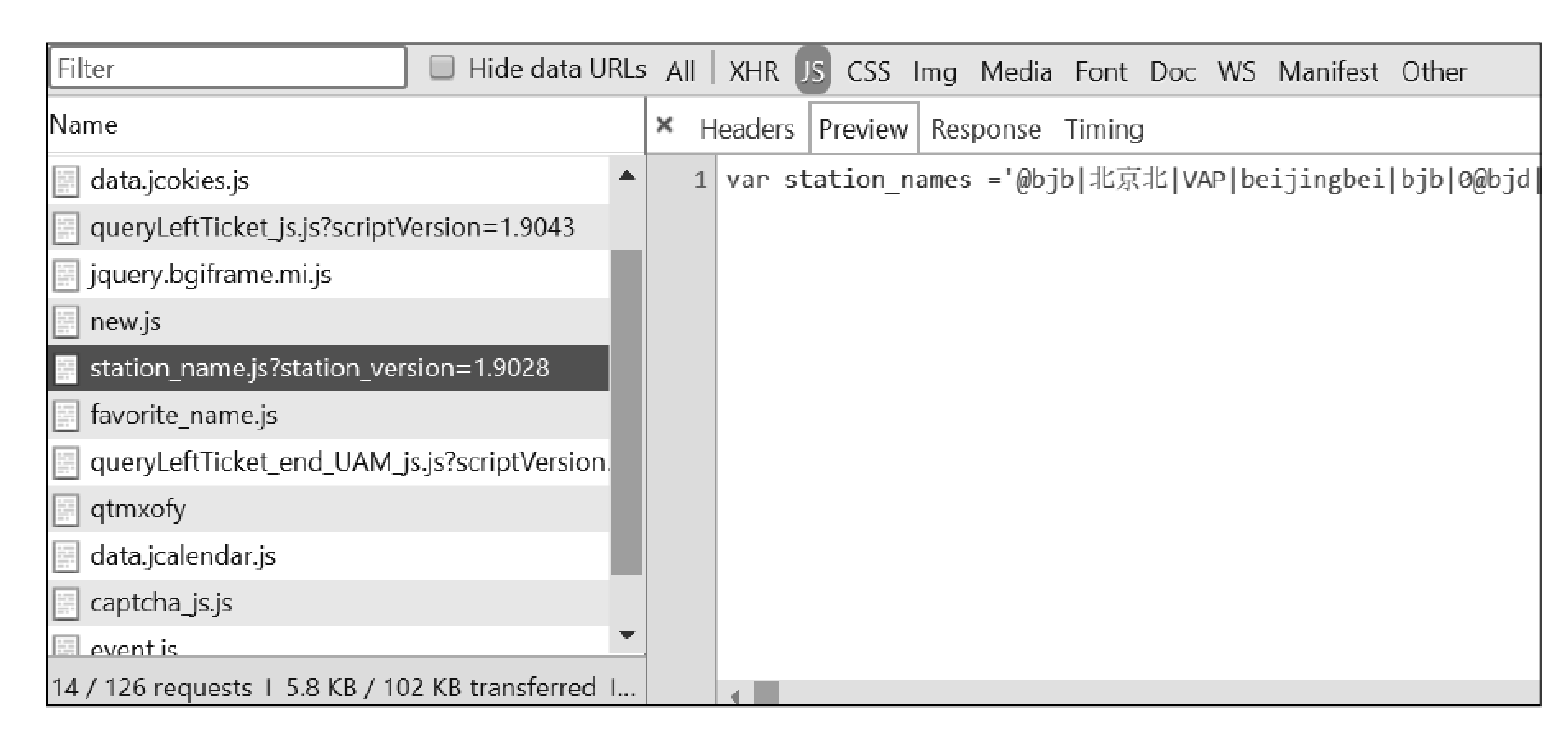


图 15-13 各个城市信息

观察其数据结构，发现每个城市之间以“@”为一个开始点，抽取部分内容进行分析：

```
@bjb|北京北|VAP|beijingbei|bjb|0@bjd|北京东|BOP|beijingdong|bjd|1@
bji|北京|BJP|beijing|bj|2@bjn
|北京南|VNP|beijingnan|bjn|
```

在内容中可以找到城市名称和城市英文编号，分别是“北京北—VAP”“北京东—BOP”和“北京—BJP”。每个数据之间以“|”隔开，而我们所需的数据分别是含有“@”的数据后的第一位和第二位。根据这个规律，实现代码如下：

```
import requests
def city_name():
    url = 'https://kyfw.12306.cn/otn/resources/js/framework/
station_name.js?station_version=1.9031'
    city_code = session.get(url)
    city_code_list = city_code.text.split("|")
    city_dict = {}
    for k, i in enumerate(city_code_list):
        if '@' in i:
  # 城市名作为字典的键，城市编号作为字典的值
            city_dict[city_code_list[k + 1]] = city_code_list[k + 2]
    return city_dict
```

现在得到图 15-11 的请求参数，接下来对图 15-11 返回的车次信息进行清洗，获取我们所需的数据。在实际中，每班车次存在两种情况，分别是有余票和无票。对这两种情况的数据进行分析和对比：

```
    "%2BJ1GyMIoe1W3AmIOyCC0ho%2FthJG%2F4PfGMtxEKS2Byrl9JuOTLHBVrBJfd4
0RUtRiP%2BbVdnpdTiob%0AjYQA1V
    kkqS16P5EsPeuK%2Fldya6KLswYyo%2BBwsLXQkr4i2D2tDAndyjyhOOhMZEKn%2B
NFAZaiBMRQi%0ATRBqKt8pYlNmBeu
    9lgEQdsdvMJ23SGTzzptyCwYtmEut4FFog6LPkywCZT1EfeFOO4Hp%2BU%2BF2NIk
%0AeeFN8fY%3D|
    预订|6c0000G31205|G312|IZQ|ICW|IZQ|WHN|06:28|10:54|04:26|Y|
    %2FCaZQBZdKPD7OTjFodC1%2F7y2N8jqdmZRAJH0rECZreKGar1Z|20171023|3|Q
Z|01|09|null|0||||||||||
    有|有|8||O0M090|OM9"
```

上述数据是图 15-12 G312 车次的列车信息，每个数据由“|”连接组成一条完整的车次信息。有一部分数据与图 15-12 对得上，其余的数据暂时无法确定，可能会在后续的流程中起到重要作用。我们再抽取无票的车次信息，如图 15-14 所示。

G542	广州南 武汉	09:44 14:01	04:17 当日到达	无	2	无	--	--	--	--	--	--	--	--	预订
K770	广州 武昌	10:00 23:41	13:41 当日到达	--	--	--	--	4	--	有	--	有	有	--	预订
G66	广州南 武汉	10:00 13:38	03:38 当日到达	无	无	无	--	--	--	--	--	--	--	--	预订
K1348	广州 武昌	10:06 23:23	13:17 当日到达	--	--	--	--	10	--	有	--	有	有	--	预订
G1006	广州南 武汉	10:11 14:17	04:06 当日到达	3	2	5	--	--	--	--	--	--	无	--	预订

图 15-14 无票的车次信息

```
"|预订|6c00000G6605|G66|IZQ|BXP|IZQ|WHN|10:00|13:38|03:38|N|
8zmC3adDZKJi1D3WPp2sMDr83uz91FxKN%2FYEoN1IG%2FD7AaME|20171023|3
|Q9|01|03|0|0|||||||||||无|无|无
||O0M090|OM9"
```

可以看到，G66 车次已经满座无票，分析其数据内容发现“预订”前面的数据为空，其他信息和有余票的车次信息大致相同。说明“预订”前面的数据可以区分车次是否还有余票。

无论是无票还是有余票，都是以“|”将各个信息连接起来组成一条车次信息，按照这个规律，车次信息清洗代码如下：

```
train_info_info = r.json()
train_info_dict = {}
for i in train_info_info['data']['result']:
train_info_status = i.split('|')
if train_info_status[0] != '':
        train_info_dict['secretStr'] = train_info_status[0]
        train_info_dict['train_no'] = train_info_status[2]
        train_info_dict['stationTrainCode'] = train_info_status[3]
        train_info_dict['fromStationTelecode'] = train_info_status[4]
        train_info_dict['toStationTelecode'] = train_info_status[7]
        train_info_dict['leftTicket'] = train_info_status[12]
        train_info_dict['train_location'] = train_info_status[15]
```

train_info_info 是图 15-11 返回的响应内容，train_info_info['data']['result'] 是直接定位车次信息，然后对车次信息以“|”分割，得到新的列表，通过判断“预订”前面的数据是否为空，排除无票的车次信息。

综合上述分析，将获取城市编号和获取车次信息的代码整合优化，代码如下：

```
# 获取城市编号
import requests
def city_name():
    url = 'https://kyfw.12306.cn/otn/resources/js/framework/
                station_name.js?station_version=1.9031'
    city_code = session.get(url)
    city_code_list = city_code.text.split("|")
```

```
        city_dict = {}
        for k, i in enumerate(city_code_list):
            if '@' in i:
      # 城市名作为字典的键，城市编号作为字典的值
                city_dict[city_code_list[k + 1]] = city_code_list[k + 2]
        return city_dict
    # 获取车次信息
    def train_info(train_date, query_from_station_name, query_to_
station_name):
        # 调用函数 city_name 获取城市编号
        city_dict = city_name()
        from_station = city_dict[query_from_station_name]
        to_station = city_dict[query_to_station_name]
        # 获取车次信息
        while 1:
            # 第一次请求
            url = 'https://kyfw.12306.cn/otn/leftTicket/log?
                    leftTicketDTO.train_date=
                                  %s&leftTicketDTO.from_station=%s&
                    leftTicketDTO.to_station=%s&purpose_codes=ADULT' %
                     (train_date, from_station, to_station)
            r = session.get(url)
            # 第二次请求
            url = 'https://kyfw.12306.cn/otn/leftTicket/queryA?
            leftTicketDTO.train_date=%s&leftTicketDTO.from_station=%s&
            leftTicketDTO.to_station=%s&purpose_codes=ADULT' %
            (train_date, from_station, to_station)
            r = session.get(url)
            time.sleep(2)
            if '非法请求' not in str(r.text) and '"result":[]' not in
str(r.text):
                train_info_info = r.json()
                train_info_dict = {}
                for i in train_info_info['data']['result']:
                    train_info_status = i.split('|')
                    if train_info_status[0] != '':
                        train_info_dict['secretStr'] = train_info_
```

```
status[0]
                         train_info_dict['train_no'] = train_info_
status[2]
                         train_info_dict['stationTrainCode'] = train_
info_status[3]
                         train_info_dict['fromStationTelecode'] =
train_info_status[4]
                         train_info_dict['toStationTelecode'] =
train_info_status[7]
                         train_info_dict['leftTicket'] = train_info_
status[12]
                         train_info_dict['train_location'] = train_
info_status[15]
                         return train_info_dict
    if __name__=='__main__':
        session = requests.session()
        username = '13435423143'
        password = 'XXXXXXXX'
        login_info = login(username,password)
        # 判断是否登录成功
        if login_info:
              train_date = '2017-10-23'
              query_from_station_name = '广州'
              query_to_station_name = '武汉'
              train_info_dict = train_info(train_date,query_from_
station_name,query_to_station_name)
```

函数 train_info() 定义三个参数：

（1）train_date 为出发时间，日期格式为 YYYY-MM-dd。

（2）query_from_station_name 为出发地，以城市中文名作为函数参数，如“广州”。

（3）query_to_station_name 为目的地，以城市中文名作为函数参数，如“武汉”。

函数整体的开发逻辑大致如下：

（1）调用函数 city_name() 获取城市编号，将出发地和目的地的中文转换成英文编号。

（2）使用 while 循环获取车次信息，目的是保证车次信息获取成功，因为在浏览器上每次查询车票不一定会返回车次信息，循环的作用相当于用户不停地单击“查询”按钮。每次循环设置延时 2 秒，这个等待时间是为了防止程序访问太过频繁而被网站认为是机器人。

（3）判断第二次请求所返回的响应内容是不是车次信息，若是，则获取第一条有余票的车次信息并返回，反之一直循环发送请求，直到获取为止。

运行上述代码，结果如图 15-15 所示。

```
请输入验证码：3
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request is being made. Adding certific
  InsecureRequestWarning)
{"result_message":"验证码校验成功","result_code":"4"}
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request is being made. Adding certific
  InsecureRequestWarning)
{"result_message":"登录成功","result_code":0,"uamtk":"K3DHvIKblTwmXOn291MjlulMQI8dZrrhonvwsw52kPI921110"}
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request is being made. Adding certific
  InsecureRequestWarning)
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request is being made. Adding certific
  InsecureRequestWarning)
{"apptk":"OOhKe__-rT1OOYcuL9mOOt4Vzy8oRAmjij2Uz5wLyOUjn1110","result_code":0,"result_message":"验证通过","username":"黄永祥"}
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request is being made. Adding certific
  InsecureRequestWarning)
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request is being made. Adding certific
  InsecureRequestWarning)
{'stationTrainCode': 'Z122', 'leftTicket': 'vLPdZKMYTFOZhyDbhCg6ZQaECB1tOU2pxNoTsf6vUoAgud%2FvHQF2CCFHw0g%3D', 'train_no': '630000Z12208',
```

图 15-15　车票查询结果

15.5 预订车票

完成车次查询，接下来实现车票预订功能。由于在 15.4 节中，函数 train_info() 只返回第一条有余票的车次信息，以图 15-12 为例，如果车次 G312 有余票，就直接返回该车次信息，如果满座无票，就取下一班车次信息再判断是否有余票，直到取得有余票的车次为止。

我们对图 15-12 的 G312 车次进行车票预订，单击“预订”按钮进行分析，发现单击按钮会触发一个 302 跳转，在跳转前，截取到两个 POST 请求，如图 15-16 和图 15-17 所示。

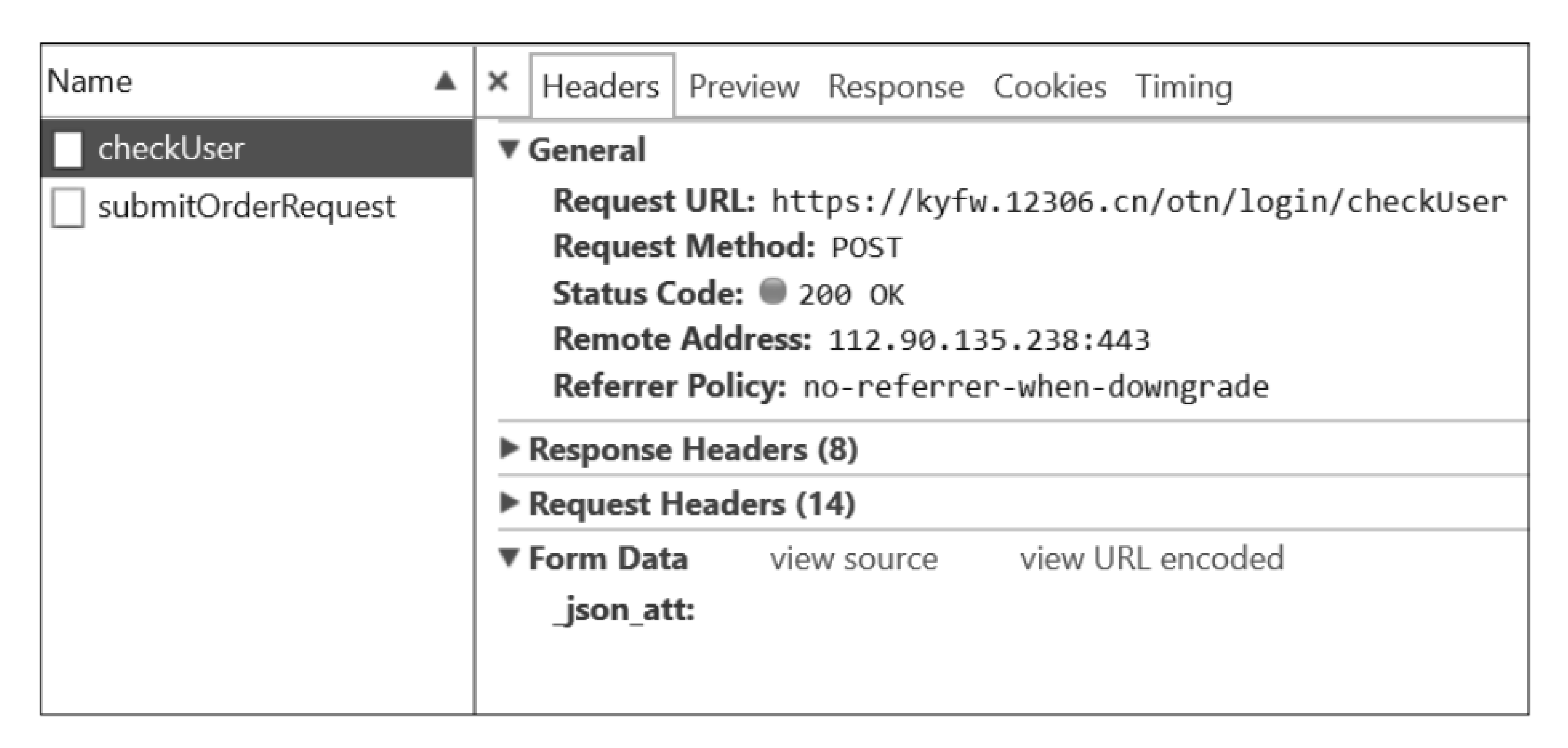


图 15-16 车票预订请求一

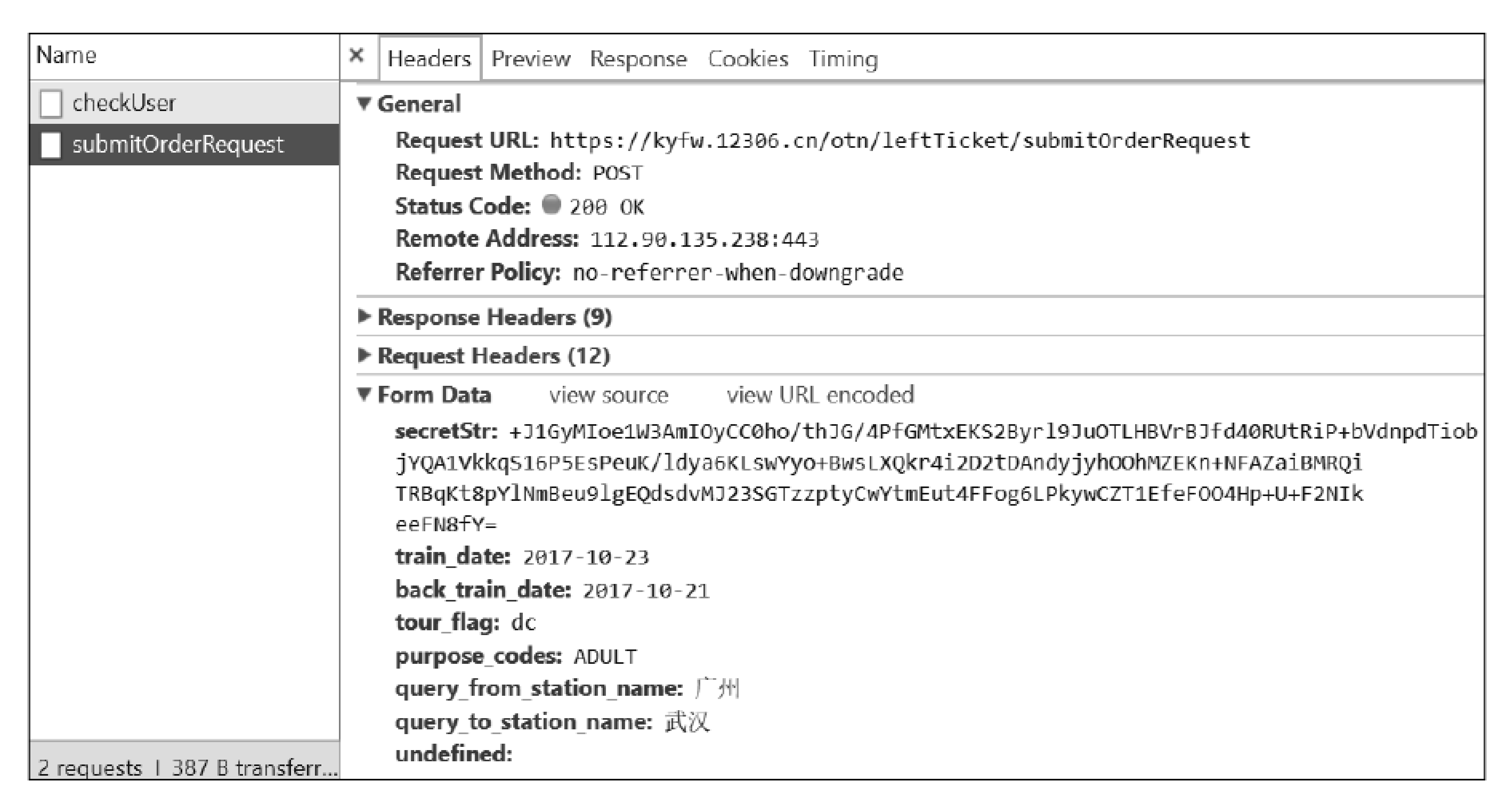


图 15-17 车票预订请求二

从图 15-16 的请求链接分析，这是用于检查用户登录状态，请求参数的数据为空。从图 15-17 的请求链接、请求参数分析得知，这是一个提交订单的请求，请求参数分析如下：

（1）train_date、query_from_station_name 和 query_to_station_name 在 15.4 节已明确知道。

（2）back_train_date 是回程的日期。如果单程订票，该参数直接取当天日期即可。

（3）tour_flag 从参数值（dc）判断，这是由“单程”拼音的首字母组成的。

（4）purpose_codes 的参数值固定不变；undefined 的参数值为空。

（5）SecretStr 是一串不规则的数据，回顾 15.4 节，在车次信息里面也含有不规则数据，猜想这两者是否一样，我们抽取 G312 车次的信息，如下所示：


```
"%2BJ1GyMIoe1W3AmIOyCC0ho%2FthJG%2F4PfGMtxEKS2Byrl9JuOTLHBVrBJfd4
0RUtRiP%2BbVdnpdTiob%0AjYQA1V
    kkqS16P5EsPeuK%2Fldya6KLswYyo%2BBwsLXQkr4i2D2tDAndyjyhOOhMZEKn%2B
NFAZaiBMRQi%0ATRBqKt8pYlNmBeu
    9lgEQdsdvMJ23SGTzzptyCwYtmEut4FFog6LPkywCZT1EfeFOO4Hp%2BU%2BF2NIk
%0AeeFN8fY%3D|
    预订|6c0000G31205|G312|IZQ|ICW|IZQ|WHN|06:28|10:54|04:26|Y|
    %2FCaZQBZdKPD7OTjFodC1%2F7y2N8jqdmZRAJH0rECZreKGar1Z|20171023|3|Q
Z|01|09|null|0|||||||||||
    有|有|8||O0M090|OM9"
```


通过对比发现两者的数据是一样的，只不过某些特殊符号经过编码处理，如“+”变成“%2B”、“/”变成“%2F”。在第 5.7 节中，urllib.parse 提供了函数对数据进行编码和解码处理。

综合上述分析，车票预订功能代码如下：

```
def train_order(secretStr, train_date, query_from_station_name,
query_to_station_name):
        # 获取当前日期
        back_train_date = datetime.datetime.now().strftime('%Y-%m-%d')
        # 用户登录检查
        url = 'https://kyfw.12306.cn/otn/login/checkUser'
        data = {
            '_json_att': ''
        }
        r = session.post(url, data=data)
        # 提交车票预订请求
        url = 'https://kyfw.12306.cn/otn/leftTicket/submitOrderRequest'
        data = {
            'secretStr': secretStr,
            'train_date': train_date,
            'back_train_date': back_train_date,
            'tour_flag': 'dc',
```

```
            'purpose_codes': 'ADULT',
            'query_from_station_name': query_from_station_name,
            'query_to_station_name': query_to_station_name,
            'undefined': ''
        }
        r = session.post(url, data=data)

    if __name__=='__main__':
        session = requests.session()
        username = '13435423143'
        password = 'xxxxxxxxxxxx'
        login_info = login(username, password)
        if login_info:
            train_date = '2017-10-23'
            query_from_station_name = '广州'
            query_to_station_name = '武汉'
            train_info_dict = train_info(train_date,
                           query_from_station_name,query_to_station_name)
            # 数据格式化处理
      secretStr=parse.unquote(train_info_dict['secretStr'])
            train_order(secretStr, train_date,
                          query_from_station_name, query_to_station_name)
```

15.6 提交订单

车票预订提交之后，从浏览器上可以看到页面发生跳转，在新的页面上需要填写乘客信息，最后提交订单，如图 15-18 所示。



图 15-18 填写乘客信息

填写好乘客信息之后，单击“提交订单”按钮，通过开发者工具捕捉请求信息，如图 15-19 和图 15-20 所示。

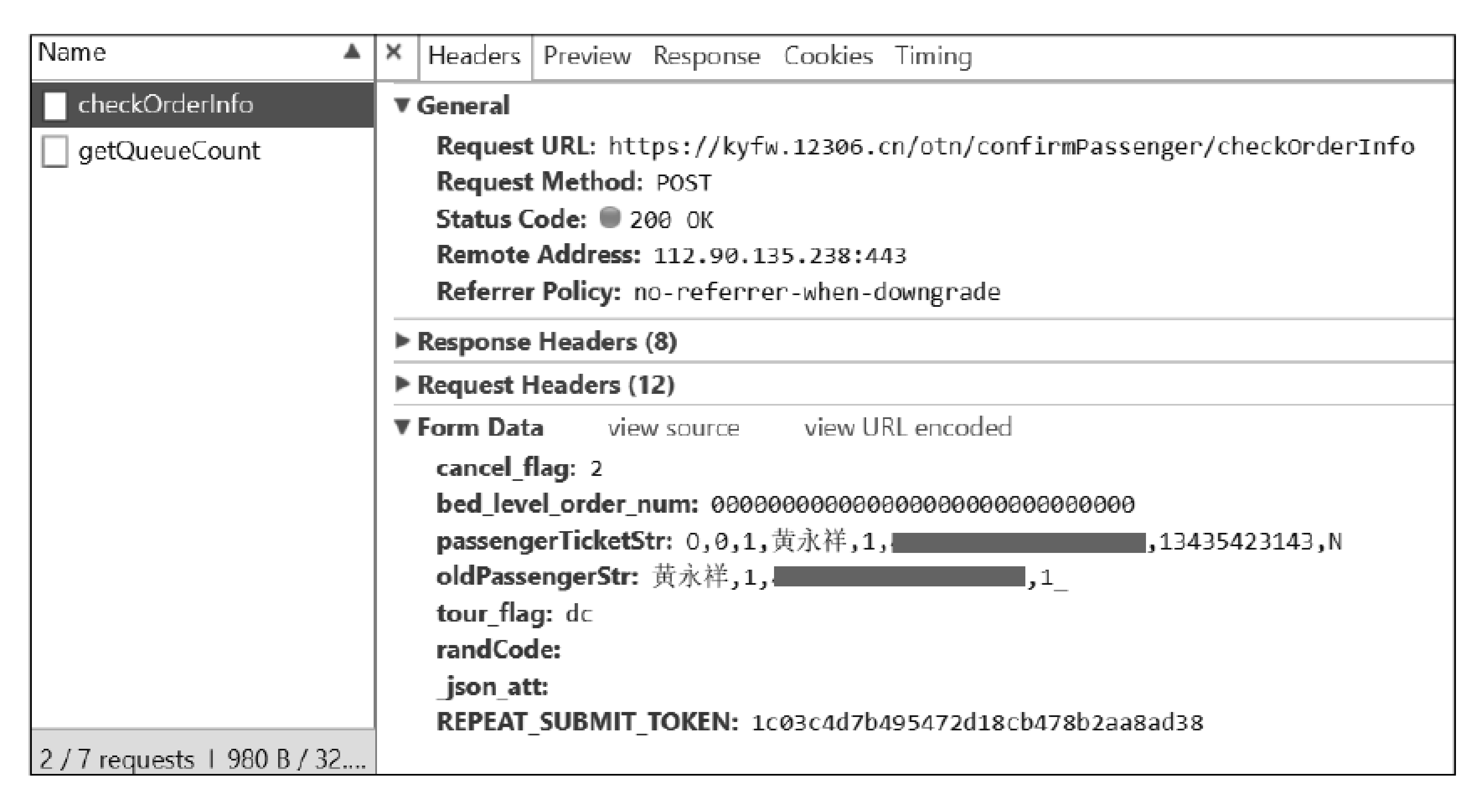


图 15-19　提交订单请求一

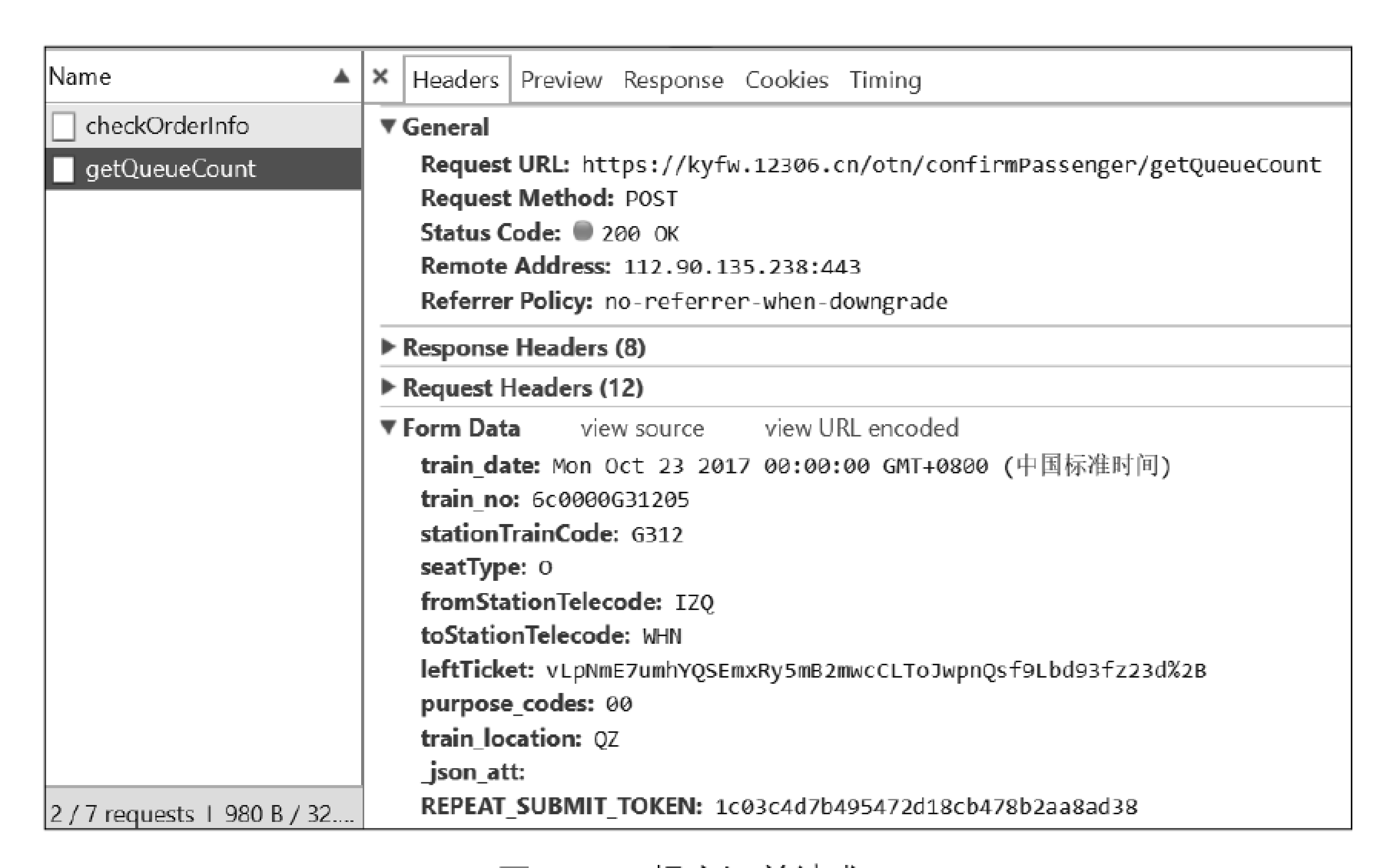


图 15-20　提交订单请求二

单击“提交订单”按钮后，Chrome 捕捉到两个 POST 请求。从图 15-18 的请求参数来看：

（1）参数 cancel_flag、bed_level_order_num、tour_flag、randCode 和 _json_att 是固定不变的。

（2）参数 REPEAT_SUBMIT_TOKEN 无法得知，可能由其他请求生成。

（3）参数 passengerTicketStr 和 oldPassengerStr 代表个人乘车信息。观察参数组成，发现每个数据之间用逗号分隔，每个数据有可能代表某个意思，为了进一步验证数据的意义，我们修改乘客信息，如图 15-21 所示。



图 15-21 验证请求参数

对比图 15-19 和图 15-21 发现，oldPassengerStr 的数据不会随着席别和票种的变化而变化，而 passengerTicketStr 会随之变化。

为了寻找 passengerTicketStr 的变化规律，多次修改乘客信息，发现变化规律如下：

（1）passengerTicketStr 前三个数字 M、0、2 的 M 和 2 分别代表一等座和儿童票，0 是固定不变的。

（2）席别编号：软卧 =4、硬座 =1、硬卧 =3、二等座 = O（字母 O）、一等座 =M、商务座 =9。

（3）票种编号：成人票 =1、儿童票 =2、学生票 =3、残军票 =4。

确定了参数 passengerTicketStr 的数据含义，同时发现一个问题，每班车次的席别信息是动态变化的，但无法确定当前车次还剩下哪些席别可供我们选择。从图 15-18 看到，席别信息是一个下拉框控件，里面含有剩余的席别信息，因此需要找出当前车次剩余的席别信息，用于构建参数 passengerTicketStr。分别从 XHR、JS 和 Doc 标签

查找席别信息，最终在 Doc 标签找到，如图 15-22 和图 15-23 所示。

```
Filter    Hide data URLs  All | XHR  JS  CSS  Img  Media  Font  Doc  WS  M
Name    ×  Headers  Preview  Response  Cookies  Timing
initDc
1399 ne':null,'id':'M','start_station_name':null,'start_time':null,'valu
1401 \u5EA7(738.50\u5143)\u6709\u7968','\u4E8C\u7B49\u5EA7(463.50\u5143)
1403 5,'minutes':28,'month':0,'seconds':0,'time':-5520000,'timezoneOffse
1 / 46 requests ...
ticket_seat_codeM  Aa  .*  738.50
```

图 15-22　查找席别信息一

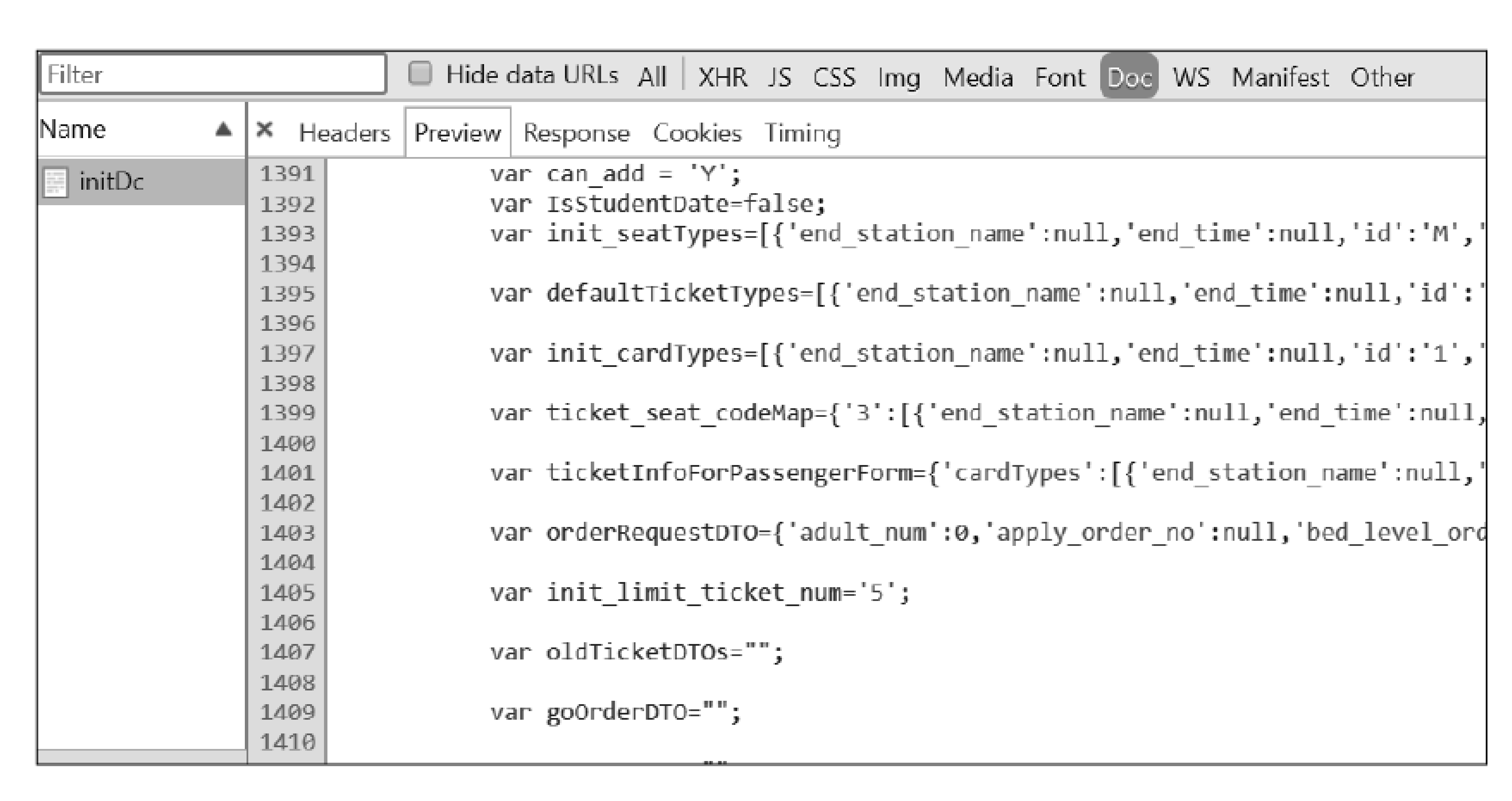


图 15-23　查找席别信息二

从图 15-22 和图 15-23 看到，由于席别的价钱具有特殊唯一性，因此可利用其特殊性来实现快速查找，发现在 Doc 的请求信息中找到剩余的席别信息。席别信息写在变量 ticket_seat_codeMap 中，以“id：X”格式存放。

再回到图 15-19 的 REPEAT_SUBMIT_TOKEN 参数，从内容中无法得知数据含义，而且数据是动态变化的，为了确认数据来源，分别从 XHR、JS 和 Doc 标签进行排查，在 Doc 标签找到数据来源，如图 15-24 所示。

```
<script>
/*<![CDATA[*/
 var ctx='/otn/';
 var globalRepeatSubmitToken = '126bd3196ce5a76fb06049a0111820ff';
 var global_lang = 'zh_CN';
 var sessionInit = '\u9EC4\u6C38\u7965';
 var isShowNotice = null;
 var CLeftTicketUrl = null;
 var isTestFlow = null;
 var isMobileCheck = null;
 var passport_appId = null;
 var passport_login = null;
 var passport_captcha = null;
 var passport_authuam = null;
 var passport_captcha_check = null;
 var passport_authclient = null;
 var passport_loginPage = null;
 var passport_okPage = null;
 var passport_proxy_captcha =  null;
 /*]]>*/
</script>
```

图 15-24 查找请求参数

可以发现，参数 REPEAT_SUBMIT_TOKEN 是 Doc 标签的 JavaScript 变量。综合上述分析，图 15-19 的实现代码如下：

```
    # 获取 Doc 标签的数据
    url = 'https://kyfw.12306.cn/otn/confirmPassenger/initDc'
    data = {
    '_json_att': '',
    }
    r = session.post(url, data=data)
    # 获取参数
    get_token = r.text.split('globalRepeatSubmitToken')[1].split(';')
[0].
    replace('=','').replace("'",'').strip()
    seat_code_str = r.text.split('ticket_seat_codeMap=')[1].
split(';')[0].strip()
    # 找出座位编号并去重
    temp_list = re.findall(r"'id':'(.+?)',",seat_code_str)
    temp_list = list(set(temp_list))
    # 获取第一个席别编号
    seatType = temp_list[0]
    # 检查订单信息
    # 构建请求参数，name-乘客姓名，identity_card-身份证号，phone_number-
电话号码，票种为成人票
```

```
    oldPassengerStr = name + ',1,' + identity_card + ',1_'
    passengerTicketStr = seatType + ',0,1,' + name + ',1,' +
identity_card + ',' + phone_number + ',N'
    url = 'https://kyfw.12306.cn/otn/confirmPassenger/checkOrderInfo'
    data = {
    'cancel_flag': '2',
    'bed_level_order_num': '000000000000000000000000000000',
    'passengerTicketStr': passengerTicketStr,
    'oldPassengerStr': oldPassengerStr,
    'tour_flag': 'dc',
    'randCode': '',
    '_json_att': '',
    'REPEAT_SUBMIT_TOKEN': get_token
    }
    r = session.post(url,data=data)
```

上述代码只实现了图 15-19 的功能，要完成订单提交，还要实现图 15-20 的请求，图 15-20 请求的参数如下：

（1）train_date、train_no、stationTrainCode、fromStationTelecode、leftTicket 和 train_location 可在 15.4 节的车次信息中获取。

（2）REPEAT_SUBMIT_TOKEN 和 seatType 可在图 15-19 实现的代码中获取。

（3）purpose_codes 和 _json_att 是固定不变的。

结合图 15-19 和图 15-20 的分析，提交订单的功能代码如下：

```
    def creat_order(name, identity_card, phone_number, train_date,
train_info_dict):
        # 获取 Doc 标签的数据
        url = 'https://kyfw.12306.cn/otn/confirmPassenger/initDc'
        data = {
            '_json_att': ''
        }
        r = session.post(url, data=data)
        # 获取参数
        key_check_isChange = r.text.split('key_check_isChange')[1].
    split(',')[0].replace(':', '').replace("'", '').strip()
```

```
        get_token = r.text.split('globalRepeatSubmitToken')[1].
    split(';')[0].replace('=', '').replace("'", '').strip()
        seat_code_str = r.text.split('ticket_seat_codeMap=')[1].
split(';')[0].strip()
        # 找出席别编号并去重
        temp_list = re.findall(r"'id':'(.+?)',", seat_code_str)
        temp_list = list(set(temp_list))
        seatType = temp_list[1]

        # 检查订单信息
        # 构建请求参数，name- 乘客姓名，identity_card- 身份证号，phone_
number- 电话号码，票种为成人票
        oldPassengerStr = name + ',1,' + identity_card + ',1_'
        passengerTicketStr = seatType + ',0,1,' + name + ',1,' +
identity_card + ',' + phone_number + ',N'
        url = 'https://kyfw.12306.cn/otn/confirmPassenger/
checkOrderInfo'
        data = {
            'cancel_flag': '2',
            'bed_level_order_num': '000000000000000000000000000000',
            'passengerTicketStr': passengerTicketStr,
            'oldPassengerStr': oldPassengerStr,
            'tour_flag': 'dc',
            'randCode': '',
            '_json_att': '',
            'REPEAT_SUBMIT_TOKEN': get_token
        }
        r = session.post(url, data=data)
        # 提交订单信息
        # train_date，train_no,stationTrainCode,fromStationTelecode,t
oStationTelecode,
        # leftTicket,train_location 来自车次信息
        # seatType 和 REPEAT_SUBMIT_TOKEN 来自 Doc 标签的数据
        # purpose_codes 和 _json_att 固定不变
        while 1:
            url = 'https://kyfw.12306.cn/otn/confirmPassenger/
getQueueCount'
```

```
            # 日期格式化处理
            check_ticket_date = train_date + ' 00:00:00'
            timeArray = time.strptime(check_ticket_date, "%Y-%m-%d
%H:%M:%S")
            date = time.strftime("%a %b %d %Y", timeArray)
            data = {
                'train_date': date + ' GMT+0800 (中国标准时间)',
                'train_no': train_info_dict['train_no'],
                'stationTrainCode': train_info_
dict['stationTrainCode'],
                'seatType': seatType,
                'fromStationTelecode': train_info_
dict['fromStationTelecode'],
                'toStationTelecode': train_info_
dict['toStationTelecode'],
                #leftTicket 进行数据格式化处理
                'leftTicket': parse.unquote(train_info_
dict['leftTicket']),
                'purpose_codes': '00',
                'train_location': train_info_dict['train_location'],
                '_json_att': '',
                'REPEAT_SUBMIT_TOKEN': get_token
            }
            r = session.post(url, data=data)
            print(r.text)
            # 判断请求是否成功
            if '系统繁忙，请稍后重试' not in str(r.text):
                break
    if __name__=='__main__':
        session = requests.session()
        username = '13435423143'
        password = 'xxxxx'
        login_info = login(username,password)
        if login_info:
            train_date = '2017-10-23'
            query_from_station_name = '广州'
            query_to_station_name = '武汉'
```

```
            train_info_dict = train_info(train_date,
                  query_from_station_name,query_to_station_name)
            secretStr = parse.unquote(train_info_dict['secretStr'])
            train_order(secretStr, train_date, query_from_station_
name, query_to_station_name)
            name = '黄永祥'
            identity_card = 'xxxxxxxxxxxxx'
            phone_number = '13435423143'
            creat_order(name, identity_card, phone_number, train_
date, train_info_dict)
```

15.7 生成订单

用户提交订单之后，下一步是确认订单，在网页上单击“提交订单”按钮，会弹出信息核对窗口，主要供用户确认乘车信息和乘客的个人信息，如图 15-25 所示。



图 15-25 信息核对

当用户信息核对无误后，单击“确认”按钮，订单就会自动生成，同时在开发者工具捕捉到两个请求信息，如图 15-26 所示。

从图 15-26 的请求参数分析，参数分为三种类型：

（1）已明确的参数，在前面的章节已分析说明，如 passengerTicketStr、oldPassengerStr、leftTicket、train_location 和 REPEAT_SUBMIT_TOKEN。

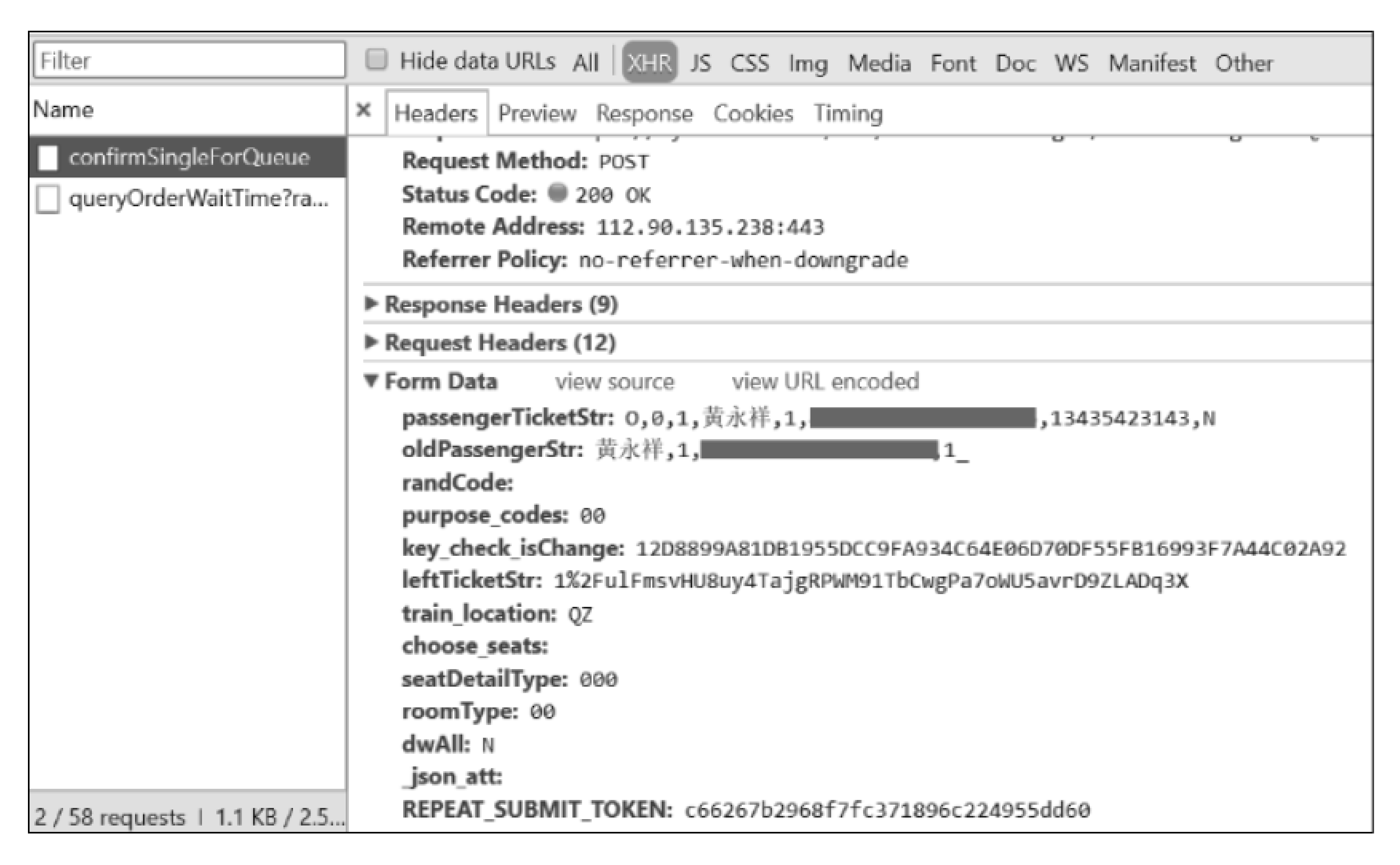


图 15-26　确认订单

（2）参数值是固定不变的，如 randCode、purpose_codes、seatDetailType、roomType、dwAll 和 _json_att。

（3）参数无法明确，如 choose_seats 和 key_check_isChange。

从参数 choose_seats 的命名分析，代表选座信息。在图 15-25 中，用户确认订单之前，还可以选择座位位置，座位以 A ～ F 命名，如果参数值为空，就默认为网站自动选座；如果参数值为 A ～ F 中的某个值，就说明车票的座位是由用户自行选择的。

参数 key_check_isChange 无法确定，要找出该参数的来源，首先分析这个请求是否由单击图 15-25 的“确认”按钮所触发，而图 15-25 的信息核对窗口是单击图 15-18 的“提交订单”按钮所产生的，在这两个过程里面，网页没有发生刷新，新增的请求信息如图 15-26 所示。这就说明，参数 key_check_isChange 可能来自于图 15-18 全部请求信息中的某个请求。因此，我们分别从 XHR、JS 和 Doc 标签查找参数，通过快捷查找，在 Doc 标签中找出参数 key_check_isChange，如图 15-27 所示。



图 15-27 查找请求参数

根据上述分析，订单生成代码如下：

```
url = 'https://kyfw.12306.cn/otn/confirmPassenger/confirmSingleForQueue'
data = {
'passengerTicketStr': passengerTicketStr,
'oldPassengerStr': oldPassengerStr,
'randCode': '',
'purpose_codes': '00',
'key_check_isChange': key_check_isChange,
'leftTicketStr': train_info_dict['leftTicket'],
'train_location': train_info_dict['train_location'],
'choose_seats': '',
'seatDetailType': '000',
'roomType': '00',
'dwAll': 'N',
'_json_att': '',
'REPEAT_SUBMIT_TOKEN': get_token
}
r = session.post(url,data=data)
print(r.text)
```

由于此功能与 15.6 节的关联较多，因此在此不再定义新的函数，将此功能直接添加到 15.6 节的函数 creat_order() 中，代码如下：

```
    def creat_order(name, identity_card, phone_number, train_date,
train_info_dict):
        # 获取 Doc 标签的数据
        url = 'https://kyfw.12306.cn/otn/confirmPassenger/initDc'
        data = {
            '_json_att': ''
        }
        r = session.post(url, data=data)
        # 获取参数
        key_check_isChange = r.text.split('key_check_isChange')[1].
split(',')[0].replace(':', '').replace("'", '').strip()
        get_token = r.text.split('globalRepeatSubmitToken')[1].
split(';')[0].replace('=', '').replace("'", '').strip()
        seat_code_str = r.text.split('ticket_seat_codeMap=')[1].
split(';')[0].strip()
        # 找出席别编号并去重
        temp_list = re.findall(r"'id':'(.+?)',", seat_code_str)
        temp_list = list(set(temp_list))
        seatType = temp_list[1]

        # 检查订单信息
        # 构建请求参数，name- 乘客姓名，identity_card- 身份证号，phone_
number- 电话号码，票种为成人票
        oldPassengerStr = name + ',1,' + identity_card + ',1_'
        passengerTicketStr = seatType + ',0,1,' + name + ',1,' +
identity_card + ',' + phone_number + ',N'
        url = 'https://kyfw.12306.cn/otn/confirmPassenger/
checkOrderInfo'
        data = {
            'cancel_flag': '2',
            'bed_level_order_num': '000000000000000000000000000000',
            'passengerTicketStr': passengerTicketStr,
            'oldPassengerStr': oldPassengerStr,
            'tour_flag': 'dc',
            'randCode': '',
            '_json_att': '',
            'REPEAT_SUBMIT_TOKEN': get_token
```

```
        }
        r = session.post(url, data=data)
        # 提交订单信息
        # train_date, train_no,stationTrainCode,fromStationTelecode,t
oStationTelecode,
        # leftTicket、train_location 来自车次信息
        # seatType 和 REPEAT_SUBMIT_TOKEN 来自 Doc 标签的数据
        # purpose_codes 和 _json_att 固定不变
        while 1:
            url = 'https://kyfw.12306.cn/otn/confirmPassenger/
getQueueCount'
            # 日期格式化处理
            check_ticket_date = train_date + ' 00:00:00'
            timeArray = time.strptime(check_ticket_date, "%Y-%m-%d
%H:%M:%S")
            date = time.strftime("%a %b %d %Y", timeArray)
            data = {
                'train_date': date + ' GMT+0800 (中国标准时间)',
                'train_no': train_info_dict['train_no'],
                'stationTrainCode': train_info_
dict['stationTrainCode'],
                'seatType': seatType,
                'fromStationTelecode': train_info_
dict['fromStationTelecode'],
                'toStationTelecode': train_info_
dict['toStationTelecode'],
                #leftTicket 进行数据格式化处理
                'leftTicket': parse.unquote(train_info_
dict['leftTicket']),
                'purpose_codes': '00',
                'train_location': train_info_dict['train_location'],
                '_json_att': '',
                'REPEAT_SUBMIT_TOKEN': get_token
            }
            r = session.post(url, data=data)
            print(r.text)
            # 判断请求是否成功
```

```
            if '系统繁忙，请稍后重试' not in str(r.text):
                break
        # 生成订单
        url = 'https://kyfw.12306.cn/otn/confirmPassenger/
confirmSingleForQueue'
        data = {
            'passengerTicketStr': passengerTicketStr,
            'oldPassengerStr': oldPassengerStr,
            'randCode': '',
            'purpose_codes': '00',
            'key_check_isChange': key_check_isChange,
            'leftTicketStr': train_info_dict['leftTicket'],
            'train_location': train_info_dict['train_location'],
            'choose_seats': '',
            'seatDetailType': '000',
            'roomType': '00',
            'dwAll': 'N',
            '_json_att': '',
            'REPEAT_SUBMIT_TOKEN': get_token
        }
        r = session.post(url, data=data)
        print(r.text)
```

15.8 本章小结

本章介绍了 12306 抢票爬虫的编写技巧，整个项目的要点总结如下：

1. 项目实现的爬虫功能

（1）验证码验证。

（2）用户登录与验证。

（3）查询车票。

（4）预订车票。

（5）提交订单。

（6）生成订单。

2. 5 个函数的功能和使用

- login()：用户验证和登录，将验证码验证和用户登录与验证合并在该函数中。
- city_name()：获取城市的编号，将城市名称转换为城市的英文编号。
- train_info()：查询车次，并调用 city_name()。
- train_order()：预订车票，主要生成订单信息。
- creat_order()：填写订单信息并提交确认，将提交订单和生成订单功能合并在该函数中。

3. 项目整体代码

```
import requests
import time
import datetime
import re
from urllib import parse

# 用户登录
def login(username, password):
    # 坐标参考：40,40,114,35,192,39,257,36,42,115,119,107,185,124,2
72,117
    code_list = {
        '1': '40,40,',
        '2': '114,35,',
        '3': '192,39,',
        '4': '257,36,',
        '5': '42,115,',
        '6': '119,107,',
        '7': '185,124,',
        '8': '272,117'
    }
    # 请求头
    headers = { 'User-Agent':
                  'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/
537.36 '
                  '(KHTML, like Gecko) Chrome/63.0.3218.0 Safari/
537.36',
                  'Referer':
```

```
                'https://kyfw.12306.cn/otn/login/init'}

url = 'https://kyfw.12306.cn/passport/captcha/captcha-image?
        login_site=E&module=login&rand=sjrand'
# 忽略证书验证
r = session.get(url, headers=headers, verify=False)
# 下载验证码图片
f = open('code.png','wb')
f.write(r.content)
f.close()
# 输入验证码图片位置，多个验证码用英文逗号分开
code=input("请输入验证码：")
get_code = ''
for i in code.split(','):
    # 根据输入每组图片的组号获取对应的坐标位置
    get_code += code_list[i]
# 验证码校验
data={
    'answer':get_code,
    'login_site':'E',
    'rand':'sjrand'
}
url = 'https://kyfw.12306.cn/passport/captcha/captcha-check'
r = session.post(url, data=data)
print(r.text)
if '验证码校验失败' not in str(r.text):
    # 用户登录
    url = 'https://kyfw.12306.cn/passport/web/login'
    data = {
        'username': username,
        'password': password,
        'appid': 'otn'
    }
    r = session.post(url, data=data)
    print(r.text)
    if '密码输入错误' not in str(r.text):
        #登录验证第一次请求
        url = 'https://kyfw.12306.cn/passport/web/auth/uamtk'
```

```
                data = {
                    'appid': 'otn'
                }
                r = session.post(url, data=data)
                # 登录验证第二次请求
                newapptk = r.json()['newapptk']
                url = 'https://kyfw.12306.cn/otn/uamauthclient'
                data = {
                    'tk': newapptk
                }
                r=session.post(url, data=data)
                print(r.text)
                return True
            else:
                return False
        return False

    # 获取城市编号
    def city_name():
        url = 'https://kyfw.12306.cn/otn/resources/js/framework/
station_name.js?
                station_version=1.9031'
        city_code = session.get(url)
        city_code_list = city_code.text.split("|")
        city_dict = {}
        for k, i in enumerate(city_code_list):
            if '@' in i:
                # 城市名作为字典的键，城市英文编号作为字典的值
                city_dict[city_code_list[k + 1]] = city_code_list[k + 2]
        return (city_dict)

    # 获取车次信息
    def train_info(train_date, query_from_station_name, query_to_
station_name):
        # 调用函数 city_name 获取城市编号
        city_dict = city_name()
        from_station = city_dict[query_from_station_name]
        to_station = city_dict[query_to_station_name]
```

```
        # 获取车次信息
        while 1:
            # 第一次请求
            url = 'https://kyfw.12306.cn/otn/leftTicket/
log?leftTicketDTO.train_date=%s&
                leftTicketDTO.from_station=%s&leftTicketDTO.to_
station=%s&purpose_codes=ADULT'
                % (train_date, from_station, to_station)
            r = session.get(url)
            # 第二次请求
            url = 'https://kyfw.12306.cn/otn/leftTicket/
queryA?leftTicketDTO.train_date=%s&
                leftTicketDTO.from_station=%s&leftTicketDTO.to_
station=%s&purpose_codes=ADULT'
                % (train_date, from_station, to_station)
            r = session.get(url)
            time.sleep(2)
            if '非法请求' not in str(r.text) and '"result":[]' not in
str(r.text):
                train_info_info = r.json()
                train_info_dict = {}
                for i in train_info_info['data']['result']:
                    train_info_status = i.split('|')
                    if train_info_status[0] != '':
                        train_info_dict['secretStr'] = train_info_
status[0]
                        train_info_dict['train_no'] = train_info_
status[2]
                        train_info_dict['stationTrainCode'] = train_
info_status[3]
                        train_info_dict['fromStationTelecode'] =
train_info_status[4]
                        train_info_dict['toStationTelecode'] = train_
info_status[7]
                        train_info_dict['leftTicket'] = train_info_
status[12]
                        train_info_dict['train_location'] = train_
```

```
info_status[15]
                        return train_info_dict

    # 预订车票
    def train_order(secretStr, train_date, query_from_station_name,
query_to_station_name):
        # 获取当前日期
        back_train_date = datetime.datetime.now().strftime('%Y-%m-
%d')
        # 用户登录检查
        url = 'https://kyfw.12306.cn/otn/login/checkUser'
        data = {
            '_json_att': ''
        }
        r = session.post(url, data=data)
        # 提交车票预订请求
        url = 'https://kyfw.12306.cn/otn/leftTicket/
submitOrderRequest'
        data = {
            'secretStr': secretStr,
            'train_date': train_date,
            'back_train_date': back_train_date,
            'tour_flag': 'dc',
            'purpose_codes': 'ADULT',
            'query_from_station_name': query_from_station_name,
            'query_to_station_name': query_to_station_name,
            'undefined': ''
        }
        r = session.post(url, data=data)

    # 生成订单
    def creat_order(name, identity_card, phone_number, train_date,
train_info_dict):
        # 获取 Doc 标签的数据
        url = 'https://kyfw.12306.cn/otn/confirmPassenger/initDc'
        data = {
            '_json_att': ''
        }
```

```
        r = session.post(url, data=data)
        # 获取参数
        key_check_isChange = r.text.split('key_check_isChange')[1].
split(',')[0].
                                     replace(':', '').replace("'", '').
strip()
        get_token = r.text.split('globalRepeatSubmitToken')[1].
split(';')[0].
                              replace('=', '').replace("'", '').strip()
        seat_code_str = r.text.split('ticket_seat_codeMap=')[1].
split(';')[0].strip()
        # 找出席别编号并去重
        temp_list = re.findall(r"'id':'(.+?)',", seat_code_str)
        temp_list = list(set(temp_list))
        seatType = temp_list[1]

        # 检查订单信息
        # 构建请求参数，name：乘客姓名，identity_card：身份证号，phone_
number：电话号码，票种为成人票
        oldPassengerStr = name + ',1,' + identity_card + ',1_'
        passengerTicketStr = seatType + ',0,1,' + name + ',1,' +
identity_card + ',' +
                                         phone_number + ',N'
        url = 'https://kyfw.12306.cn/otn/confirmPassenger/
checkOrderInfo'
        data = {
            'cancel_flag': '2',
            'bed_level_order_num': '000000000000000000000000000000',
            'passengerTicketStr': passengerTicketStr,
            'oldPassengerStr': oldPassengerStr,
            'tour_flag': 'dc',
            'randCode': '',
            '_json_att': '',
            'REPEAT_SUBMIT_TOKEN': get_token
        }
        r = session.post(url, data=data)
        # 提交订单信息
```

```
        # train_date, train_no,stationTrainCode,fromStationTelecode,t
oStationTelecode,
        # leftTicket,train_location 来自车次信息
        # seatType 和 REPEAT_SUBMIT_TOKEN 来自 Doc 标签的数据
        # purpose_codes 和 _json_att 固定不变
        while 1:
            url = 'https://kyfw.12306.cn/otn/confirmPassenger/
getQueueCount'
            # 日期格式化处理
            check_ticket_date = train_date + ' 00:00:00'
            timeArray = time.strptime(check_ticket_date, "%Y-%m-%d
%H:%M:%S")
            date = time.strftime("%a %b %d %Y", timeArray)
            data = {
                'train_date': date + ' GMT+0800 (中国标准时间)',
                'train_no': train_info_dict['train_no'],
                'stationTrainCode': train_info_
dict['stationTrainCode'],
                'seatType': seatType,
                'fromStationTelecode': train_info_
dict['fromStationTelecode'],
                'toStationTelecode': train_info_
dict['toStationTelecode'],
                #leftTicket 进行数据格式化处理
                'leftTicket': parse.unquote(train_info_
dict['leftTicket']),
                'purpose_codes': '00',
                'train_location': train_info_dict['train_location'],
                '_json_att': '',
                'REPEAT_SUBMIT_TOKEN': get_token
            }
            r = session.post(url, data=data)
            print(r.text)
            # 判断请求是否成功
            if '系统繁忙，请稍后重试' not in str(r.text):
                break
        # 生成订单
```

```
        url = 'https://kyfw.12306.cn/otn/confirmPassenger/
confirmSingleForQueue'
        data = {
            'passengerTicketStr': passengerTicketStr,
            'oldPassengerStr': oldPassengerStr,
            'randCode': '',
            'purpose_codes': '00',
            'key_check_isChange': key_check_isChange,
            'leftTicketStr': train_info_dict['leftTicket'],
            'train_location': train_info_dict['train_location'],
            'choose_seats': '',
            'seatDetailType': '000',
            'roomType': '00',
            'dwAll': 'N',
            '_json_att': '',
            'REPEAT_SUBMIT_TOKEN': get_token
        }
        r = session.post(url, data=data)
        print(r.text)

    if __name__ == '__main__':
        session = requests.session()
        # 网站账号密码
        username = '13435423143'
        password = 'xxxxxx'
        login_info = login(username, password)
        if login_info:
            train_date = 'YYYY-mm-DD'
            query_from_station_name = '广州'
            query_to_station_name = '武汉'
            train_info_dict = train_info(train_date, query_from_
station_name, query_to_station_name)
            secretStr = parse.unquote(train_info_dict['secretStr'])
            train_order(secretStr, train_date, query_from_station_
name, query_to_station_name)
            # 乘客信息
            name = '黄永祥'
```

```
            identity_card = 'xxxxxxxxxxxxxxxxx'
            phone_number = '13435423143'
            creat_order(name, identity_card, phone_number, train_
date, train_info_dict)
```

上述代码运行结果如图 15-28 所示。

```
{"result_message":"验证码校验成功","result_code":"4"}
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request is being made. Adding certificate verification
  InsecureRequestWarning)
{"result_message":"登录成功","result_code":0,"uamtk":"[illegible]"}
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request is being made. Adding certificate verification
  InsecureRequestWarning)
{"apptk":"[illegible]","result_code":0,"result_message":"验证通过","username":"[illegible]"}
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request is being made. Adding certificate verification
  InsecureRequestWarning)
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request is being made. Adding certificate verification
  InsecureRequestWarning)
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request is being made. Adding certificate verification
  InsecureRequestWarning)
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request is being made. Adding certificate verification
  InsecureRequestWarning)
{"validateMessagesShowId":"_validatorMessage","status":true,"httpstatus":200,"data":"N","messages":[],"validateMessages":{}}
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request is being made. Adding certificate verification
  InsecureRequestWarning)
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request is being made. Adding certificate verification
  InsecureRequestWarning)
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request is being made. Adding certificate verification
  InsecureRequestWarning)
F:\Python\lib\site-packages\urllib3\connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request is being made. Adding certificate verification
{"validateMessagesShowId":"_validatorMessage","status":true,"httpstatus":200,"data":{"count":"0","ticket":"11","op_2":"false","countT":"0","op_1":"false"}
  InsecureRequestWarning)
{"validateMessagesShowId":"_validatorMessage","status":true,"httpstatus":200,"data":{"submitStatus":true},"messages":[],"validateMessages":{}}
```

图 15-28 程序运行结果

从图 15-28 最后的数据看到 "httpstatus":200,"data":{"submitStatus":true}，代表购票已成功。用户可以在“我的 12306 →未完成订单”中查看已生成的订单内容，最后的付款流程需要用户自行完成，此时整个购票流程真正完成。

4. 进一步完善的建议

在本项目中只是简单地实现一个抢票过程，但遇到春运期间的抢票，程序的稳定性需要进一步修改和完善，下面列出几条值得完善的建议：

（1）增加车次可选择功能，将查询出来的车次的发车时间、时长等信息提供给用户自行选择。

（2）判断订单生成状态，对生成失败的订单进行相应处理。

（3）异常处理机制，因为网站的稳定性一直是饱受争议的问题，所以要完善异常处理机制，确保出现异常的时候能及时处理，提高程序的稳定性。

第 16 章

项目实战：玩转微博

16.1 分析说明

接触过微博的读者都知道，一些热门的微博有很多转发、评论和点赞，而且博主有很庞大的粉丝数。这些高数据都离不开营销手段，在庞大的数据中有多少是真实数据不为人知，但可以肯定的是，这些数据肯定有水分存在。那么这些有水分的数据是如何产生的呢？这就是本章讲述的重点。

本章主要实现的功能如下。

- weibo_login.py：微博用户登录，同时也是程序运行文件。
- weibo_verify_code.py：第三方平台 API，实现验证码识别。
- weibo_collect.py：根据关键字搜索并采集热门微博。

- weibo_send.py：发布微博。
- weibo_follow.py：关注用户。
- weibo_forward.py：微博点赞和转发评论。
- data.csv：存储采集数据。
- 文件夹 video 和 image：分别存储采集的视频和图片。

16.2 用户登录

进入微博首页，我们发现微博大部分功能都要用户登录才能使用。那么爬取微博的第一步就是实现用户登录。在 Chrome 浏览器对微博的登录机制进行分析，在浏览器中输入 http://weibo.com/，打开开发者工具，捕捉首页的请求信息，如图 16-1 所示。

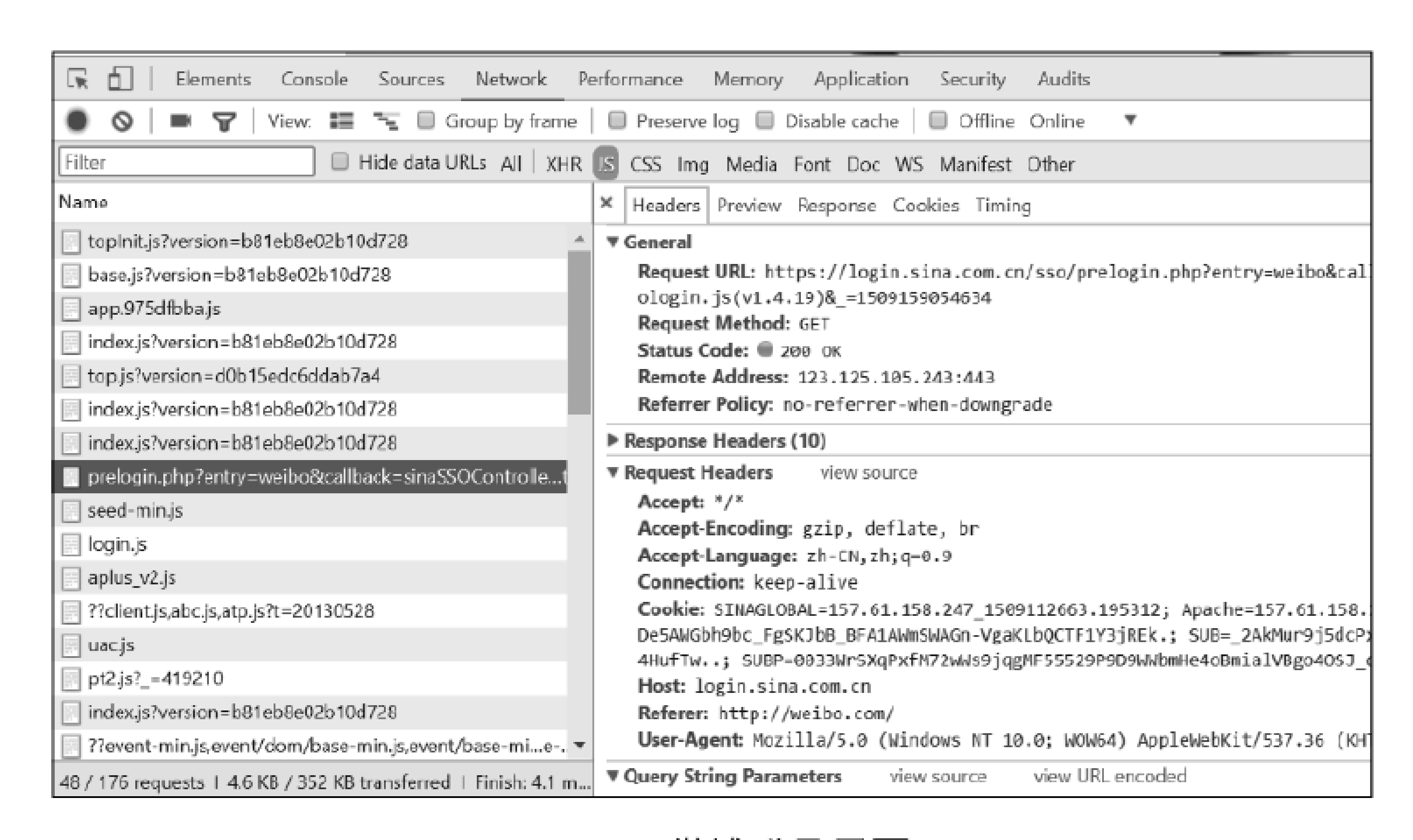


图 16-1 微博登录界面

在开发者工具里分别查看 XHR、JS 和 Doc 标签的请求信息：

（1）Doc 标签有 4 个请求信息，请求信息的响应内容都是 HTML，主要是网页的布局和一些 JavaScript 脚本信息。

（2）XHR 标签有一个 POST 的请求信息，对该信息的请求链接、请求参数和响应内容进行分析，该请求信息与登录信息没有太大关联。

（3）JS 标签有多个请求信息，大多数请求都是 JavaScript 脚本内容，查看每一个请求信息，发现其中一个请求较为特殊，如图 16-2 所示。

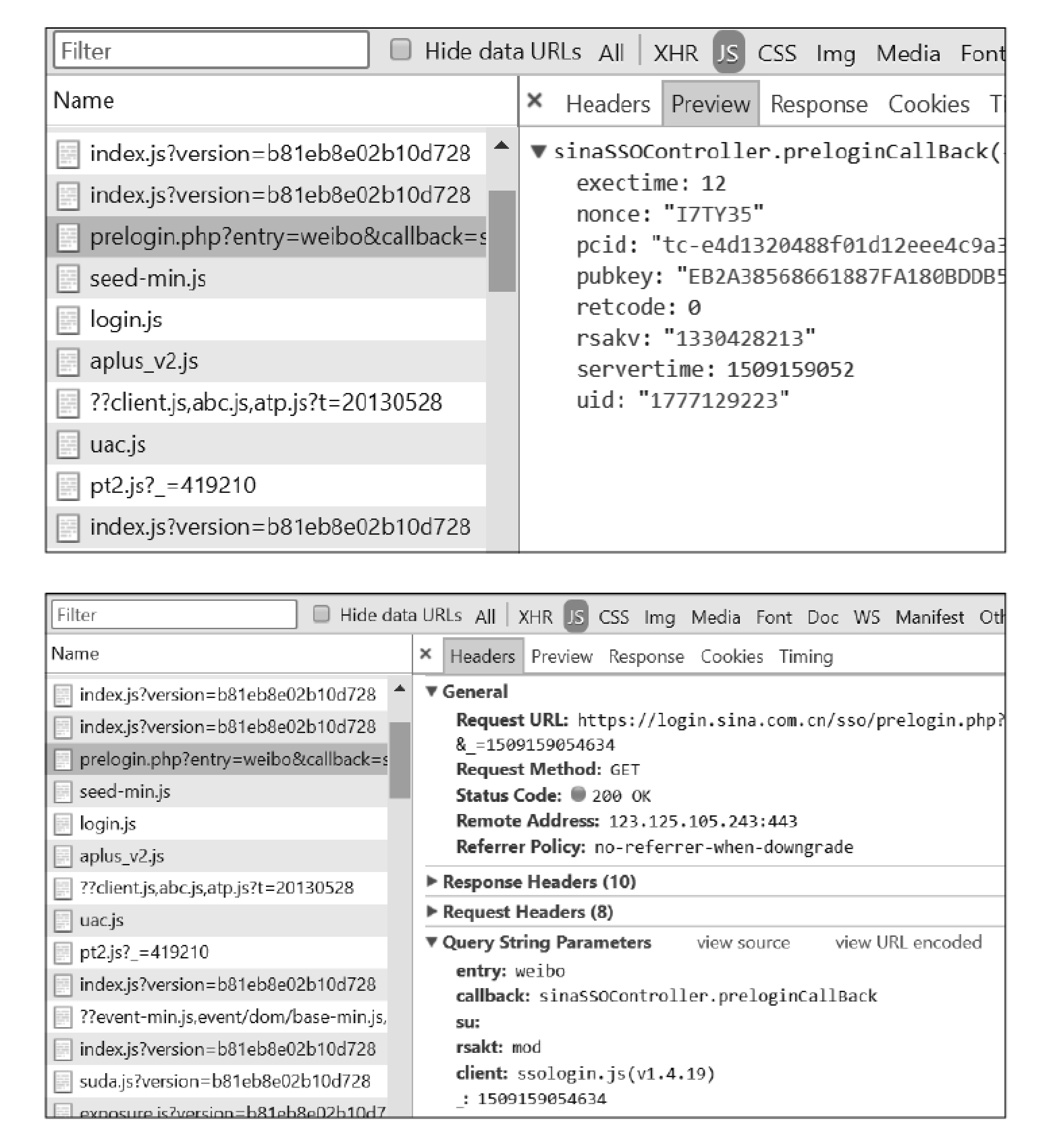


图 16-2 微博登录分析

根据图 16-2 分析请求参数和响应内容，含义如下。

- su：代表用户账号，一般以 su 或 username 命名。
- 1509159054634：以“150”开头的数字大多数是时间戳。
- rsakt 和 rsakv：无法确定这两个参数代表的含义，但两者都含有 rsa，rsa 是一个加密方法。参数值可能经过加密处理。
- pubkey：中文翻译为公共密钥，从这个参数可知，某些数据肯定做过加密处理，大多数是对账号、密码做加密处理。

通过简单分析，我们知道在用户登录之前会触发一个准备登录（prelogin）请求，该请求中包含一些加密信息。也就是说，在实现登录功能之前，先要对上述请求信息发送请求，获取其响应内容的加密信息后，才能进行下一步用户登录。实现代码如下：

```
    import requests
    import time
    def get_server_data(su):
        # 构建 URL
        prelogin_url = 'https://login.sina.com.cn/sso/prelogin.
php?entry=weibo&callback=
                              sinaSSOController.preloginCallBack&su=%s&r
sakt=mod&
                              client=ssologin.js(v1.4.19)&_=%s' %(su,
str(int(time.time() * 1000)))
        pre_data_res = session.get(prelogin_url, headers=headers ,
proxies=proxies)
        # 将响应内容转换为字典格式
        sever_data = eval(pre_data_res.content.decode("utf-8").
                        replace("sinaSSOController.preloginCallBack", ''))
        return sever_data
    if __name__ == "__main__":
        # 构造请求头
        agent = 'Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0)
Gecko/20100101 Firefox/41.0'
        headers = {
            'User-Agent': agent
        }
        # 代理 IP，防止同一 IP 登录多个不同微博账号
        proxies = {}
        # 新建会话
        session = requests.session()
        # 用户账号
        su = '13435423143'
        sever_data = get_server_data(su)
```

现在得到了登录的加密信息，但还不知道具体使用了哪些加密方法，我们知道网站对数据加密一般都在前端完成加密处理，然后将加密的数据发送到网站后台，在后台再对数据解密处理并返回响应，这样的方法可以提高数据在发送传输时的安全性。

根据上述原理，我们可以在请求信息中找出具体的加密方法，数据加密主要以 JavaScript 实现，对 JS 标签里的各个 JS 文件进行分析，找到实现加密功能的 JS 文件，如图 16-3 所示。



图 16-3　微博登录加密方法

通过分析图 16-3 中请求信息的响应内容（JavaScript 代码）可以发现：

（1）用户账号主要使用 base64 方式加密。

（2）密码是使用 RSA 加密的，加密密钥是图 16-2 中的 servertime、nonce 和 pubkey。

根据上述分析，账号、密码使用不同的加密方式，对此分别对两者定义不同的函数，代码如下：

```
import urllib
import base64
import rsa
import binascii
# 账号加密
def get_su(username):
    # 使用 urllib.parse.quote_plus 对 email 地址或手机号码的特殊符号编码处理
    # 然后使用 base64 加密
    username_quote = urllib.parse.quote_plus(username)
    username_base64 = base64.b64encode(username_quote.encode("utf-8"))
    return username_base64.decode("utf-8")

# 密码加密，servertime、nonce、pubkey 是来自图 16-2 的数据
def get_password(password, servertime, nonce, pubkey):
```

```
        rsaPublickey = int(pubkey, 16)
        # 创建公钥
        key = rsa.PublicKey(rsaPublickey, 65537)
        # 拼接明文
        message = str(servertime) + '\t' + str(nonce) + '\n' +
str(password)
        message = message.encode("utf-8")
        # 加密
        passwd = rsa.encrypt(message, key)
        # 将加密信息转换为 16 进制
        passwd = binascii.b2a_hex(passwd)
        return passwd
```



rsa 模块是第三方库，可使用 pip install rsa 安装。

完成用户的账号、密码加密处理后，最后一步就是实现用户登录，在浏览器中输入账号、密码，单击“登录”按钮，分析开发者工具捕捉到的请求信息，如图 16-4 所示。

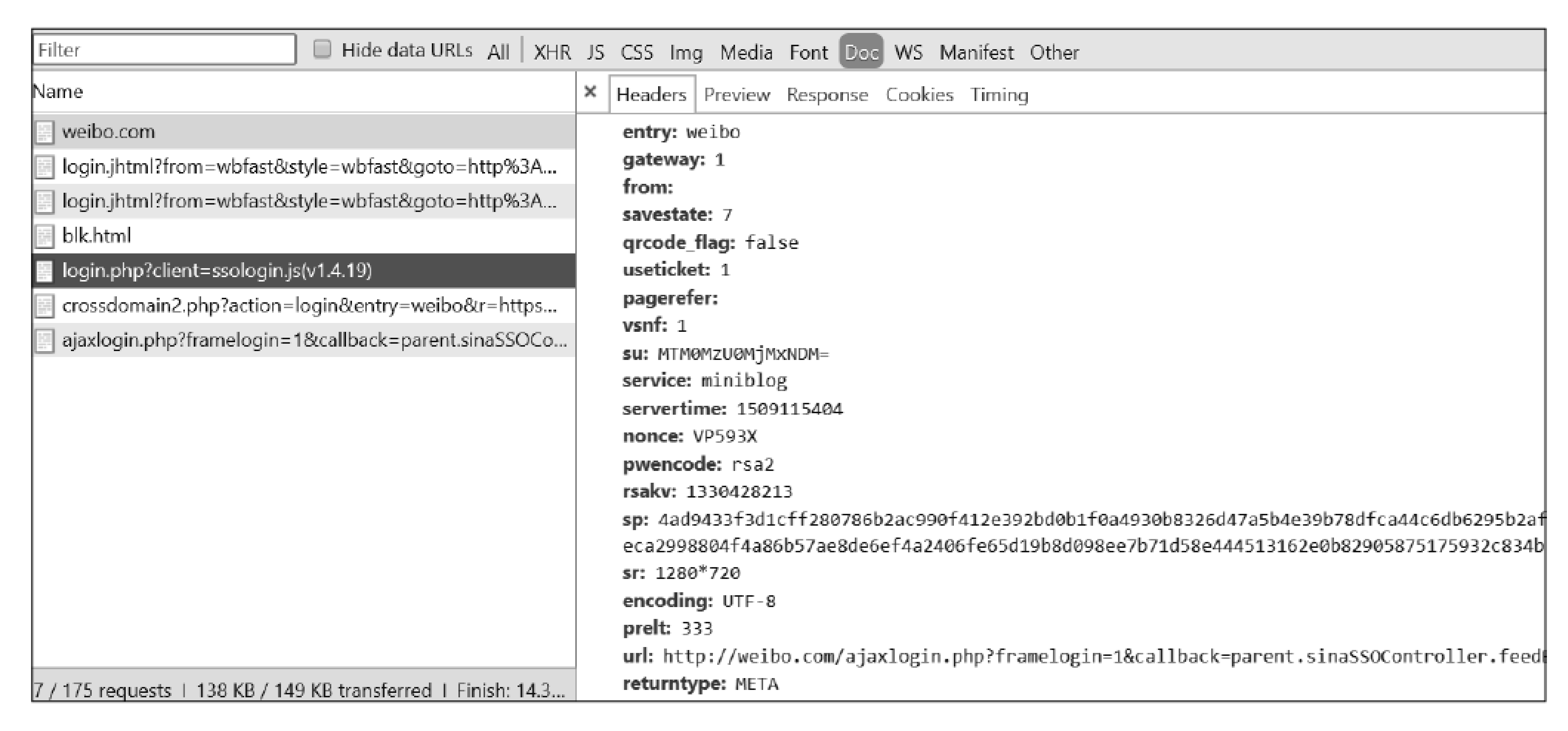


图 16-4 微博用户登录

从请求参数可知，su、sp、servertime、nonce 和 rsakv 是动态变化的，其他参数都是固定不变的。而 servertime、nonce 和 rsakv 可以在图 16-2 中直接获取，su 和 sp 分别是加密后的账号和密码。用户登录代码实现如下：

```
import time
import base64
import rsa
import binascii
import requests
import re,urllib
def login(username, password):
    # 获取 servertime、nonce、rsakv、su 和 sp
    su = get_su(username)
    sever_data = get_server_data(su)
    servertime = sever_data["servertime"]
    nonce = sever_data['nonce']
    rsakv = sever_data["rsakv"]
    pubkey = sever_data["pubkey"]
    sp = get_password(password, servertime, nonce, pubkey)
    # 构建请求参数
    data = {
        'entry': 'weibo',
        'gateway': '1',
        'from': '',
        'savestate': '7',
        'useticket': '1',
        'pagerefer': "http://login.sina.com.cn/sso/logout.php?ent
ry=miniblog&r=http%3A%2F%2Fweibo.com%2Flogout.php%3Fbackurl",
        'vsnf': '1',
        'su': su,
        'service': 'miniblog',
        'servertime': servertime,
        'nonce': nonce,
        'pwencode': 'rsa2',
        'rsakv': rsakv,
        'sp': sp,
        'sr': '1366*768',
        'encoding': 'UTF-8',
        'prelt': '115',
        'url': 'http://weibo.com/ajaxlogin.php?framelogin=1&callb
ack=parent.sinaSSOController.feedBackUrlCallBack',
```

```
            'returntype': 'META'
            }
        # 用户登录
        url = 'http://login.sina.com.cn/sso/login.
php?client=ssologin.js(v1.4.18)'
        login_page = session.post(url, data=data,proxies=proxies)
        print(login_page.text) ①
    if __name__ == "__main__":
        # 请求头
        agent = 'Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0)
Gecko/20100101 Firefox/41.0'
        headers = {
            'User-Agent': agent
        }
        # 代理 IP，防止同一 IP 登录多个不同微博账号
        proxies = {}
        # 新建会话
        session = requests.session()
        login('13435423143','xxxxxxxxxx')
```

运行上述代码，结果如图 16-5 所示。

```
<meta http-equiv="Content-Type" content="text/html; charset=GBK" />
<title>新浪通行证</title>


<script charset="utf-8" src="https://i.sso.sina.com.cn/js/ssologin.js"></script>
</head>
<body>
正在登录 ...
<script>
try{sinaSSOController.setCrossDomainUrlList({"retcode":0,"arrURL":["https:\/\/passport.97973.com\/sso\/crossdomain?action=login&savestate=15407099
        catch(e){
            var msg = e.message;
            var img = new Image();
            var type = 1;
            img.src = 'https://login.sina.com.cn/sso/debuglog?msg=' + msg +'&type=' + type;
        }try{sinaSSOController.crossDomainAction('login',function(){location.replace('https://passport.weibo.com/wbsso/login?ssosavestate=15407099
        catch(e){
            var msg = e.message;
            var img = new Image();
            var type = 2;
            img.src = 'https://login.sina.com.cn/sso/debuglog?msg=' + msg +'&type=' + type;
        }
</script>
</body>
</html>
```

图 16-5 用户登录响应内容一

响应内容是一个 HTML 格式的数据，在 HTML 内容中无法得知是否登录成功，因为在数据中无法获取用户的信息。但细心分析可知，HTML 内容中有“location.replace”，这是一个页面跳转的功能，以此作为突破口，可以尝试访问跳转的链接，看能否在这个链接中获取用户信息。在上述代码中的①处添加以下代码：

```
login_loop = (login_page.content.decode("GBK"))
# 网页跳转 URL，获取用户信息
pa = r'location\.replace\([\'"](.*?)[\'"]\)'
loop_url = re.findall(pa, login_loop)[0]
login_index = session.get(loop_url, proxies=proxies)
print(login_index.text) ②
```

再次运行代码，结果如图 16-6 和图 16-7 所示。

({"result":true,"userinfo":{"uniqueid":"1777129223","userid":null,"displayname":null,"userdomain":"?wvr=5&lf=reg"}})

图 16-6 用户登录响应内容二

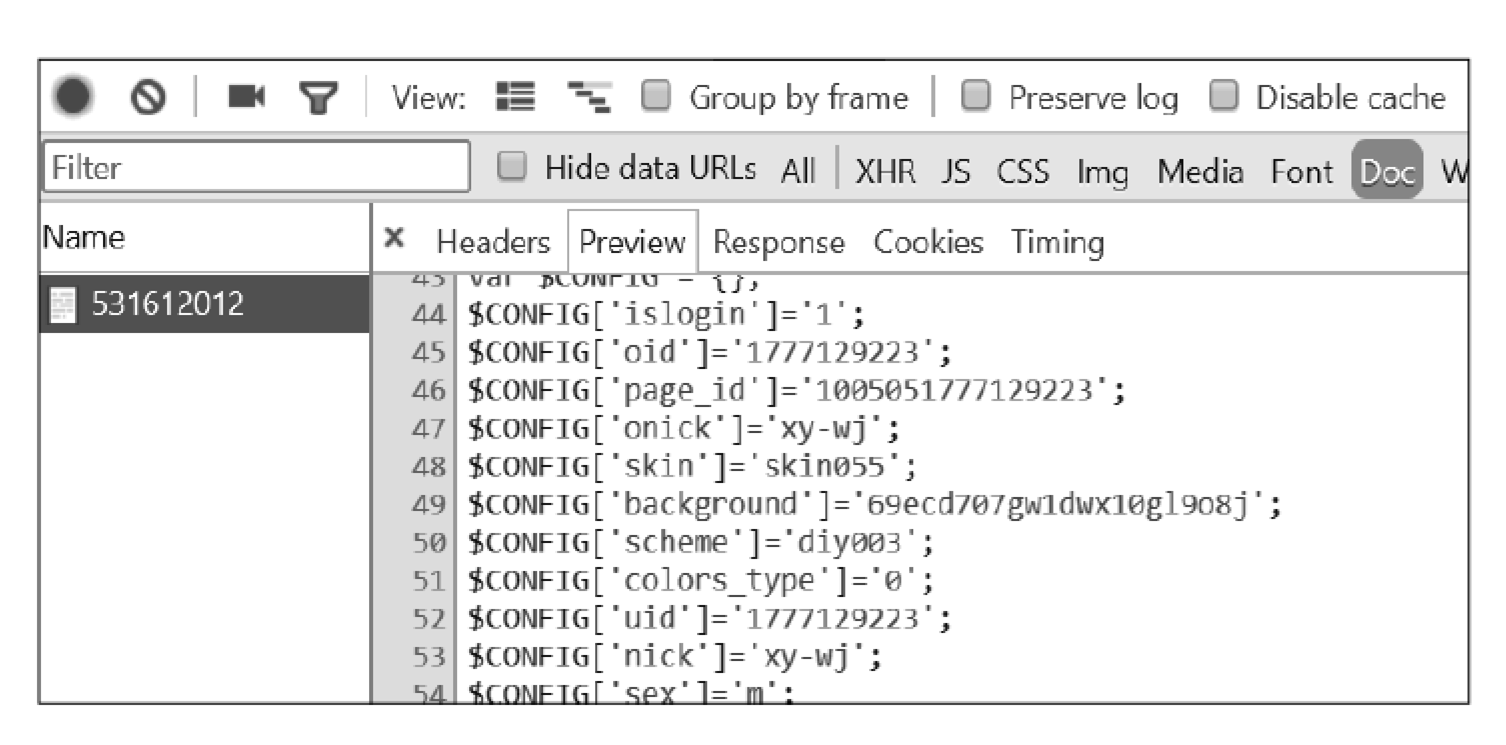


图 16-7 微博用户信息

图 16-7 是在网页上查看的微博用户首页信息。对比图 16-6 和图 16-7，图 16-6 说明用户已成功登录，其中 userinfo 代表用户信息，观察 userinfo 的数据，发现 uniqueid 等于图 16-7 中的 uid 和 oid，因此根据 uniqueid 获取用户首页信息，在上述代码的②处加入以下代码：

```
uuid = login_index.text
uuid_pa = r'"uniqueid":"(.*?)"'
uuid_res = re.findall(uuid_pa, uuid, re.S)[0]
# 根据 uniqueid 构建微博首页的 URL
web_weibo_url = "http://weibo.com/%s" % uuid_res
weibo_page = session.get(web_weibo_url, proxies=proxies)
response = weibo_page.text
```

```
    person_info = {}
    if '$CONFIG' in response:
    person_info['nick'] = response.split("$CONFIG['nick']='")[1].
split("';")[0]
    person_info['watermark'] = response.
split("$CONFIG['watermark']='")[1].split("';")[0]
    person_info['location'] = response.split("$CONFIG['location']='")
[1].split("';")[0]
    person_info['uid'] = response.split("$CONFIG['uid']='")[1].
split("';")[0]
    person_info['domain'] = response.split("$CONFIG['domain']='")[1].
split("';")[0]
    person_info['oid'] = response.split("$CONFIG['oid']='")[1].
split("';")[0]
    print(' 登录成功，你的用户名为：' + person_info['nick'])
```

综合上述已实现的功能，本节完整的代码如下：

```
    import requests
    import time
    import urllib
    import base64
    import rsa
    import binascii
    import re
    # 登录前准备
    def get_server_data(su):
        # 构建 URL
        prelogin_url = 'https://login.sina.com.cn/sso/prelogin.
php?entry=weibo&callback=
              sinaSSOController.preloginCallBack&su=%s&rsakt=mod&client=
     ssologin.js(v1.4.19)&_=%s' %(su, str(int(time.time() * 1000)))
        pre_data_res = session.get(prelogin_url, headers=headers,
proxies=proxies)
        # 将响应内容转换为字典格式
        sever_data = eval(pre_data_res.content.decode("utf-8").
                      replace("sinaSSOController.preloginCallBack", ''))
        return sever_data
```

```
    # 账号加密
    def get_su(username):
        # 使用 urllib.parse.quote_plus 对 email 地址或手机号码的特殊符号进行
编码处理
        # 然后使用 base64 加密
        username_quote = urllib.parse.quote_plus(username)
        username_base64 = base64.b64encode(username_quote.
encode("utf-8"))
        return username_base64.decode("utf-8")

    # 密码加密，servertime、nonce、pubkey 是来自图 16-2 的数据
    def get_password(password, servertime, nonce, pubkey):
        rsaPublickey = int(pubkey, 16)
        # 创建公钥
        key = rsa.PublicKey(rsaPublickey, 65537)
        # 拼接明文
        message = str(servertime) + '\t' + str(nonce) + '\n' +
str(password)
        message = message.encode("utf-8")
        # 加密
        passwd = rsa.encrypt(message, key)
        # 将加密信息转换为 16 进制
        passwd = binascii.b2a_hex(passwd)
        return passwd

    # 用户登录
    def login(username, password):
        # 获取 servertime、nonce、rsakv、su 和 sp
        su = get_su(username)
        sever_data = get_server_data(su)
        servertime = sever_data["servertime"]
        nonce = sever_data['nonce']
        rsakv = sever_data["rsakv"]
        pubkey = sever_data["pubkey"]
        sp = get_password(password, servertime, nonce, pubkey)
        # 构建请求参数
```

```
        data = {
            'entry': 'weibo',
            'gateway': '1',
            'from': '',
            'savestate': '7',
            'useticket': '1',
            'pagerefer': "http://login.sina.com.cn/sso/logout.
php?entry=miniblog&r=
                                    http%3A%2F%2Fweibo.com%2Flogout.
php%3Fbackurl",
            'vsnf': '1',
            'su': su,
            'service': 'miniblog',
            'servertime': servertime,
            'nonce': nonce,
            'pwencode': 'rsa2',
            'rsakv': rsakv,
            'sp': sp,
            'sr': '1366*768',
            'encoding': 'UTF-8',
            'prelt': '115',
            'url': 'http://weibo.com/ajaxlogin.
php?framelogin=1&callback=
                               parent.sinaSSOController.feedBackUrlCallBack',
            'returntype': 'META'
            }
        # 用户登录
        login_url = 'http://login.sina.com.cn/sso/login.
php?client=ssologin.js(v1.4.18)'
        login_page = session.post(login_url, data=data)
        login_loop = (login_page.content.decode("GBK"))
        # 网页跳转 URL，获取用户信息
        pa = r'location\.replace\([\'"](.*?)[\'"]\)'
        loop_url = re.findall(pa, login_loop)[0]
        login_index = session.get(loop_url)
        uuid = login_index.text
        uuid_pa = r'"uniqueid":"(.*?)"'
```

```
        uuid_res = re.findall(uuid_pa, uuid, re.S)[0]
        # 根据 uniqueid 构建微博首页 URL
        web_weibo_url = "http://weibo.com/%s" % uuid_res
        weibo_page = session.get(web_weibo_url)
        response = weibo_page.text
        person_info = {}
        if '$CONFIG' in response:
            person_info['nick'] = response.split("$CONFIG['nick']='")
[1].split("';")[0]
            person_info['watermark'] = response.
split("$CONFIG['watermark']='")[1].split("';")[0]
            person_info['location'] = response.
split("$CONFIG['location']='")[1].split("';")[0]
            person_info['uid'] = response.split("$CONFIG['uid']='")
[1].split("';")[0]
            person_info['domain'] = response.
split("$CONFIG['domain']='")[1].split("';")[0]
            person_info['oid'] = response.split("$CONFIG['oid']='")
[1].split("';")[0]
            print('登录成功，你的用户名为：' + person_info['nick'])
        else:
            print('登录失败')
        return person_info

    if __name__ == "__main__":
        # 构造请求头
        agent = 'Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0)
Gecko/20100101 Firefox/41.0'
        headers = {
            'User-Agent': agent
        }
        # 代理 IP
        proxies = {}
        # 新建会话
        session = requests.session()
        user_info = login('13435423143','xxxxxx')
```

16.3 用户登录（带验证码）

16.3 节已实现微博用户登录，如果要实现多账号批量登录，那么需要使用代理 IP 实现，否则同一个 IP 登录多个账号，账号很容易被网站查封。在使用代理 IP 登录微博时，有可能遇到验证码验证的问题，为解决验证码验证问题，我们在浏览器上设置代理 IP，如图 16-8 所示。

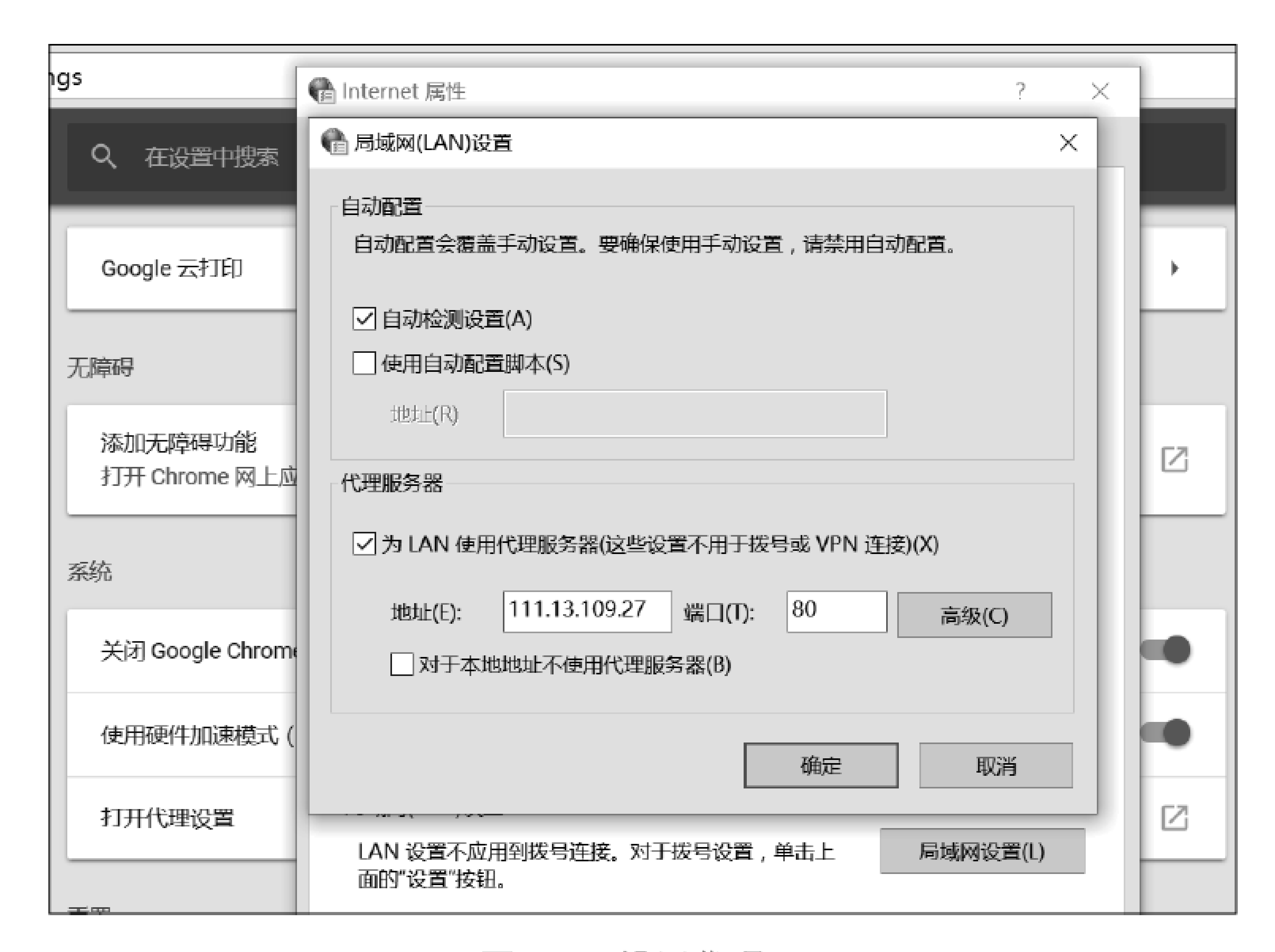


图 16-8 设置代理 IP

设置代理 IP 之后，返回微博登录界面，可以看到登录界面出现验证码，如图 16-9 所示。

打开开发者工具，查看请求信息进行分析，找到验证码图片请求信息，如图 16-10 所示。



图 16-9 带验证码微博登录



图 16-10　图片验证码请求信息

从图 16-10 中看到有三个请求参数：r 是一个随机数，生成的规律不固定；s 是一个固定数字；p 是一个不可知的数据，需要找出该数据的来源。

再分析登录前的加密信息（prelogin）是否也发生变化，如图 16-11 所示。

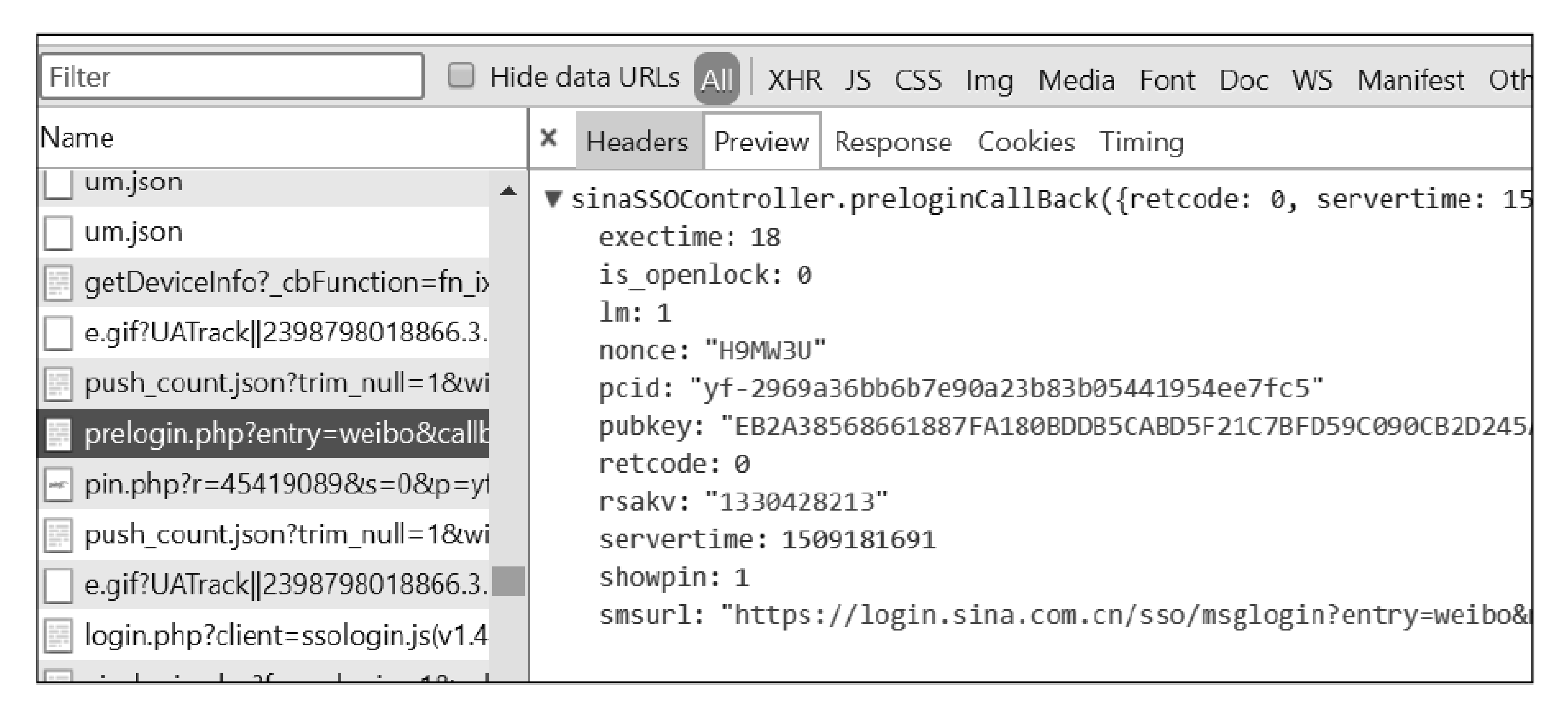


图 16-11　带验证码的登录信息

对比图 16-11 与图 16-2，发现带验证码的登录前加密信息中的数据多了 showpin、is_openlock 和 lm；再与图 16-10 对比，发现图 16-11 中 pcid 的数据与图 16-10 中 p 参数的值相同。

结合上述分析：

（1）访问加密信息，根据返回内容进行判断，如果存在 showpin、is_openlock 和 lm 数据，就说明当前登录需要验证码识别。

（2）如果存在验证码，就先下载验证码图片，再进行下一步的用户登录；否则直接执行 16.2 节的代码。

下载验证码图片的代码如下：

```
def get_img(pcid):
    url = 'https://login.sina.com.cn/cgi/pin.php?r=%s&s=0&p=%s'
      %(str(math.floor(random.random() * 100000000)),pcid)
    resp = session.get(url)
    verify_code_path = '%s.png' % (str(int(time.time() * 1000)))
    f = open(verify_code_path, 'wb')
    f.write(resp.content)
    f.close()
    return verify_code_path
```

在函数 get_img() 中，参数 pcid 由加密信息的 pcid 传递，最后函数返回的是图片的相对路径。完成验证码下载后，接着分析带验证码的登录请求，如图 16-12 所示。

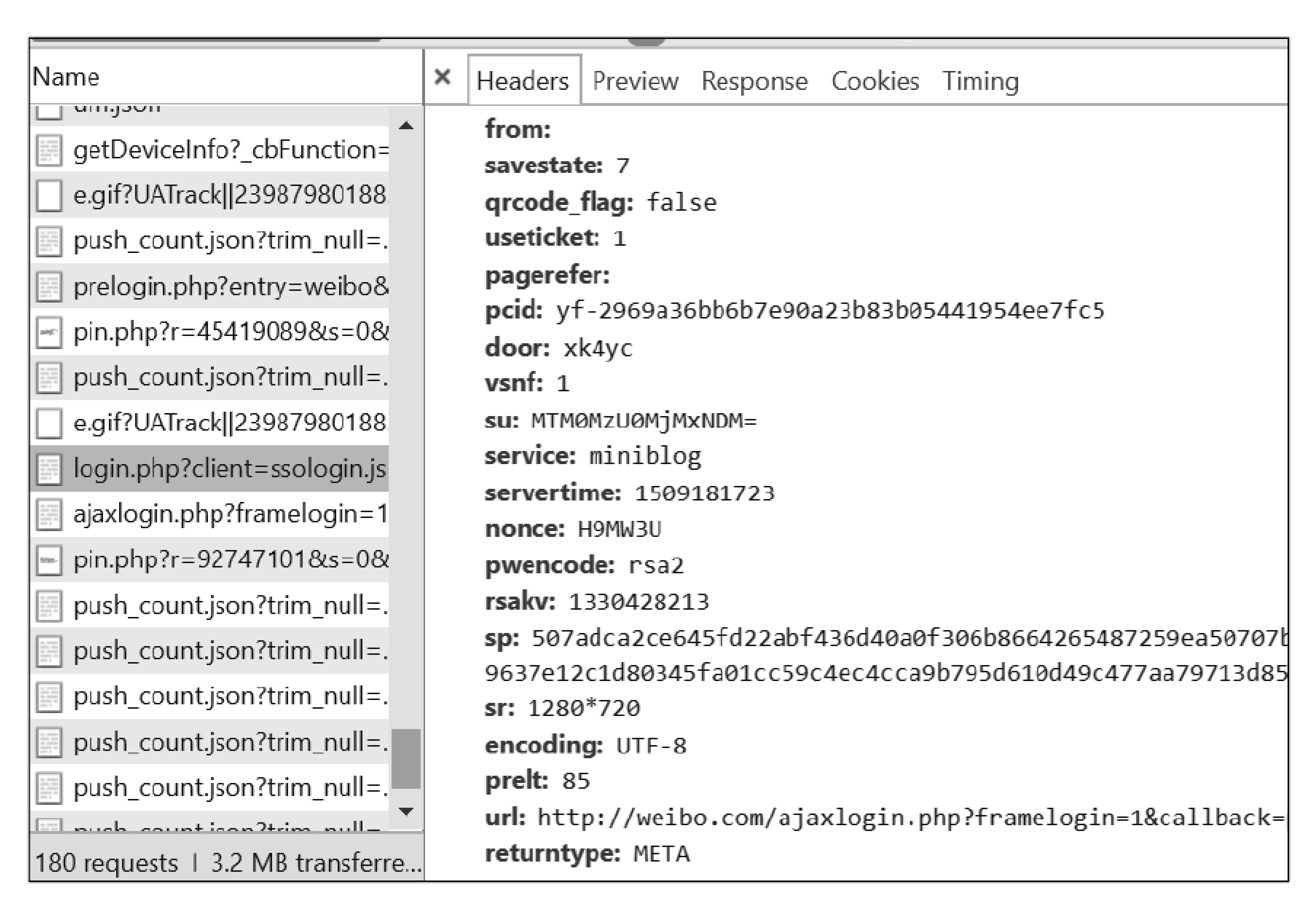


图 16-12 带验证码用户登录

从图 16-12 和图 16-4 的请求参数对比得出，图 16-12 的请求参数多了 pcid 和 door。pcid 是来自加密信息里的响应数据，door 是验证码图片内容。

根据上述分析，总结如下：

（1）如果带有验证码，加密信息的响应内容就含有 showpin、is_openlock 和 lm 数据。

（2）判断加密信息的响应内容是否需要下载验证码图片。

（3）下载图片验证码后，需要对验证码进行识别。

（4）在用户登录请求中，带验证码的登录需要添加参数 pcid 和 door。

在上面的分析要点中，目前还没解决验证码识别的问题。7.3 节讲述了第三方平台如何识别验证码，因此本项目使用第三方平台提供的 API 解决验证码识别问题，将 API 代码命名并保存在文件 weibo_verify_code.py 中。在 16.2 节的代码中加入验证码处理功能，代码如下：

```
import requests
import time
import urllib
import base64
import rsa
import binascii
import re
# 接入第三方 API 识别验证码
from weibo_verify_code import code_verificate
# 登录前准备
def get_server_data(su):
    # 构建 URL
    prelogin_url = 'https://login.sina.com.cn/sso/prelogin.
php?entry=weibo&callback=
                    sinaSSOController.preloginCallBack&su=%s&rsakt=mo
d&client=
                    ssologin.js(v1.4.19)&_=%s' %(su, str(int(time.
time() * 1000)))
        pre_data_res = session.get(prelogin_url, headers=headers,
proxies=proxies)
        # 将响应内容转换为字典格式
        sever_data = eval(pre_data_res.content.decode("utf-8").
                     replace("sinaSSOController.preloginCallBack", ''))
```

```
        return sever_data

    # 账号加密
    def get_su(username):
        # 使用 urllib.parse.quote_plus 对 email 地址或手机号码的特殊符号进行
编码处理
        # 然后使用 base64 加密
        username_quote = urllib.parse.quote_plus(username)
        username_base64 = base64.b64encode(username_quote.
encode("utf-8"))
        return username_base64.decode("utf-8")

    # 密码加密，servertime、nonce、pubkey 是来自图 16-2 的数据
    def get_password(password, servertime, nonce, pubkey):
        rsaPublickey = int(pubkey, 16)
        # 创建公钥
        key = rsa.PublicKey(rsaPublickey, 65537)
        # 拼接明文
        message = str(servertime) + '\t' + str(nonce) + '\n' +
str(password)
        message = message.encode("utf-8")
        # 加密
        passwd = rsa.encrypt(message, key)
        # 将加密信息转换为 16 进制
        passwd = binascii.b2a_hex(passwd)
        return passwd

    # 下载验证码图片
    def get_img(pcid):
        url = 'https://login.sina.com.cn/cgi/pin.php?r=%s&s=0&p=%s'
              %(str(math.floor(random.random() * 100000000)),pcid)
        resp = session.get(url)
        verify_code_path = '%s.png' % (str(int(time.time() * 1000)))
        f = open(verify_code_path, 'wb')
        f.write(resp.content)
        f.close()
        return verify_code_path
```

```
# 用户登录
def login(username, password):
    # 获取 servertime、nonce、rsakv、su 和 sp
    su = get_su(username)
    sever_data = get_server_data(su)
    servertime = sever_data["servertime"]
    nonce = sever_data['nonce']
    rsakv = sever_data["rsakv"]
    pubkey = sever_data["pubkey"]
    sp = get_password(password, servertime, nonce, pubkey)
    # 构建请求参数
    data = {
        'entry': 'weibo',
        'gateway': '1',
        'from': '',
        'savestate': '7',
        'useticket': '1',
        'pagerefer': "http://login.sina.com.cn/sso/logout.
php?entry=miniblog&r=
                             http%3A%2F%2Fweibo.com%2Flogout.
php%3Fbackurl",
        'vsnf': '1',
        'su': su,
        'service': 'miniblog',
        'servertime': servertime,
        'nonce': nonce,
        'pwencode': 'rsa2',
        'rsakv': rsakv,
        'sp': sp,
        'sr': '1366*768',
        'encoding': 'UTF-8',
        'prelt': '115',
        'url': 'http://weibo.com/ajaxlogin.
php?framelogin=1&callback=
                     parent.sinaSSOController.feedBackUrlCallBack',
        'returntype': 'META'
```

```
            }
        # 判断是否存在验证码
        if 'showpin' in sever_data.keys():
            # 添加请求参数
            pcid = sever_data['pcid']
            data['pcid'] = pcid
            # 下载验证码图片
            verify_code_path = get_img(pcid)
            # 第三方平台识别验证码
            verify_code = code_verificate(yundama_username, yundama_
password, verify_code_path)
            print(verify_code)
            data['door'] = verify_code

        # 用户登录
        login_url = 'http://login.sina.com.cn/sso/login.
php?client=ssologin.js(v1.4.18)'
        login_page = session.post(login_url, data=data)
        login_loop = (login_page.content.decode("GBK"))
        # 网页跳转 URL，获取用户信息
        pa = r'location\.replace\([\'"](.*?)[\'"]\)'
        loop_url = re.findall(pa, login_loop)[0]
        login_index = session.get(loop_url)
        uuid = login_index.text
        uuid_pa = r'"uniqueid":"(.*?)"'
        uuid_res = re.findall(uuid_pa, uuid, re.S)[0]
        # 根据 uniqueid 构建微博首页 URL
        web_weibo_url = "http://weibo.com/%s" % uuid_res
        weibo_page = session.get(web_weibo_url)
        response = weibo_page.text
        person_info = {}
        if '$CONFIG' in response:
            person_info['nick'] = response.split("$CONFIG['nick']='")
[1].split("';")[0]
            person_info['watermark'] = response.
```

```
split("$CONFIG['watermark']='")[1].split("';")[0]
            person_info['location'] = response.
split("$CONFIG['location']='")[1].split("';")[0]
            person_info['uid'] = response.split("$CONFIG['uid']='")
[1].split("';")[0]
            person_info['domain'] = response.
split("$CONFIG['domain']='")[1].split("';")[0]
            person_info['oid'] = response.split("$CONFIG['oid']='")
[1].split("';")[0]
            print('登录成功，你的用户名为：' + person_info['nick'])
        else:
            print('登录失败')
        return person_info

    if __name__ == "__main__":
        # 构造请求头
        agent = 'Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0)
Gecko/20100101 Firefox/41.0'
        headers = {
            'User-Agent': agent
        }
        # 代理 IP
        proxies = {
            'http': "http://113.214.13.1:8000/"
        }
        # 新建会话
        session = requests.session()
        # 第三方平台账号密码
        yundama_username = 'xxxxxx'
        yundama_password = 'xxxxxx'
        user_info = login('13435423143','xxxxxx')
```

程序运行结果如图 16-13 所示。

```
F:\Python\python.exe F:/微博/weibo_Software/weibo_Software/weibo_login.py
uid: 53927
balance: 1472
cid: 1573126148, result: VMYMK
VMYMK
登陆成功，你的用户名为：xy-wj

Process finished with exit code 0
```

图 16-13 带验证码的微博用户登录结果

16.4 关键字搜索热门微博

完成用户登录后，接着实现关键字搜索热门微博，该功能可以让我们及时掌握微博最新的咨询以及各个行业的动态走向，巧妙运用这个功能等于拥有了微博平台的大数据。

在浏览器中打开网站 http://s.weibo.com/，以“# 王者荣耀 #”为关键字，如图 16-14 所示。



图 16-14 微博高级搜索功能

每次搜索网页都会重新刷新一遍，说明搜索结果是由后台直接生成的。我们直接分析 Doc 标签的请求信息，如图 16-15 所示。

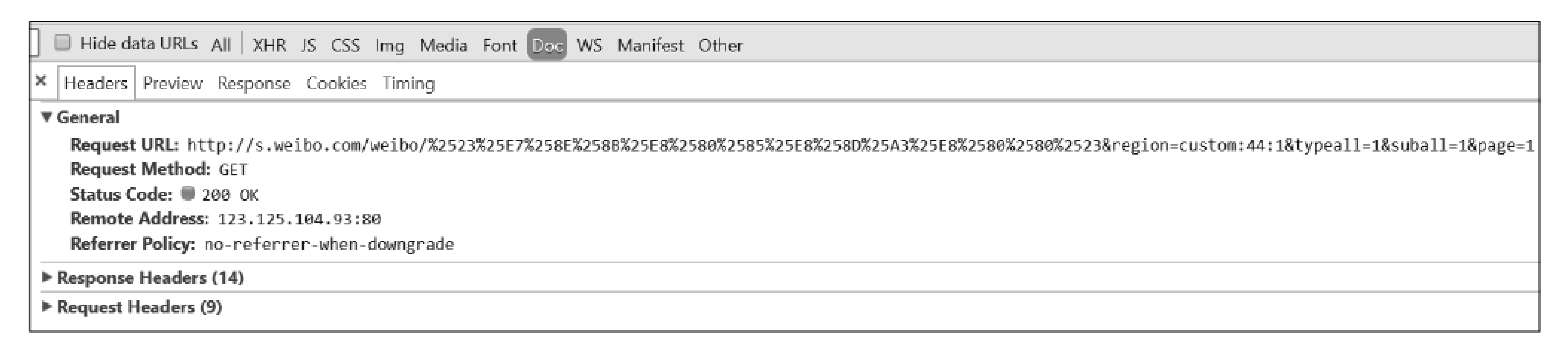


图 16-15　微博高级搜索请求信息

请求参数分析如下：

（1）URL 含有“%2523%25E7%”，说明 URL 部分数据进行了编码处理，编码内容有可能是关键字，可以使用 urllib.parse.unquote_plus 对 URL 的编码部分进行还原。

（2）region 是地区内容，44 代表广东省，1 代表广州。

（3）page 代表页数，一个关键字最多返回 50 页内容。

（4）其余的参数是固定不变的。

分析响应内容，发现网页显示的内容在响应内容中无法找到，但细心观察就会发现，网页上显示的中文内容在响应内容中都变成乱码，而且存放在 JavaScript 中，如图 16-16 所示。

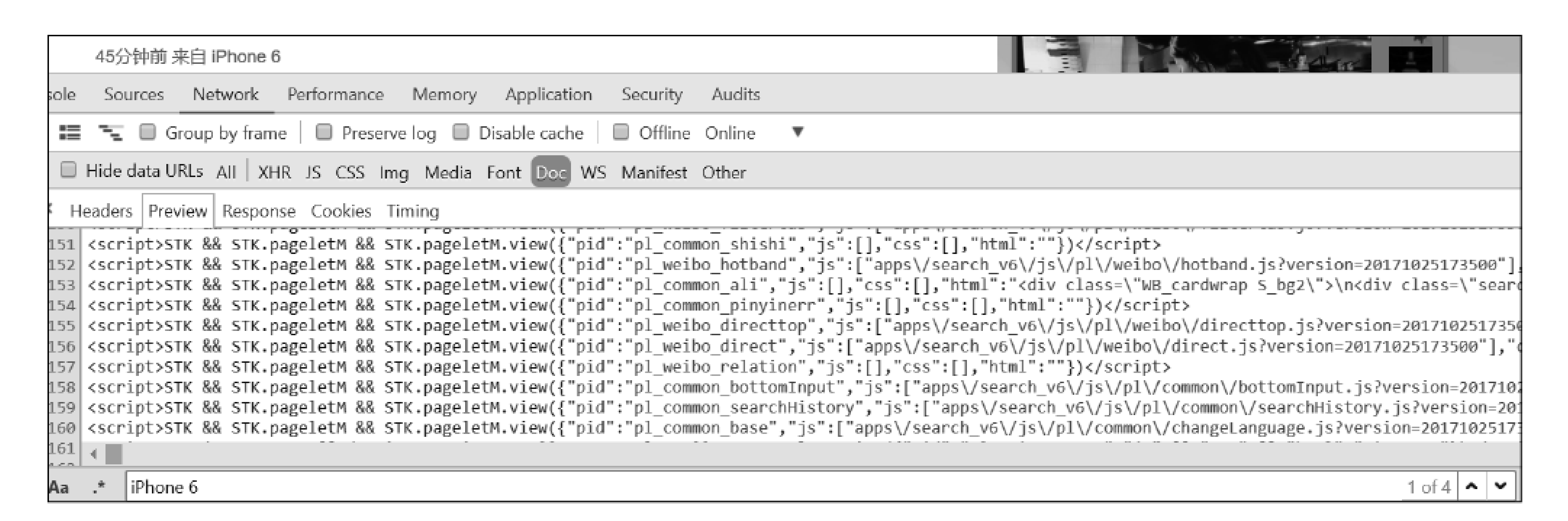


图 16-16　查找微博内容

综合分析，实现代码如下：

```
def collect(keyword, pagenumber=1):
    # 关键字编码
    keyword_change = urllib.parse.quote_plus(keyword)
    keyword_change = urllib.parse.quote_plus(keyword_change)
```

```
now = datetime.datetime.now().strftime('%Y-%m-%d')
# 构建 URL，地区默认广东省广州市
url='http://s.weibo.com/weibo/'+keyword_change+'&region=
   custom:44:1&typeall=1&suball=1&page=%s' %(str(pagenumber))
r = session.get(url)
# 解决中文乱码
get_value = r.content.decode('unicode_escape').replace('\/', '/')
f = open('data.txt', 'w' ,encoding='utf-8')
f.write(get_value)
f.close()
```

上述代码需要用户登录后才能运行，我们将搜索结果记录在“data.txt”文件中，打开文件并对内容进行分析，如图 16-17 所示。

```
<div class="face">
<a          suda-data="key=tblog_search_weibo&value=weibo_ss_4_icon"
href="http://weibo.com/lyle?refer_flag=1001030103_" title="御遠" target="_blank">
<img src="http://tvax4.sinaimg.cn/crop.0.0.996.996.50/625564f4ly8fhrp24f92mj20ro0roweq.jpg" alt="御遠" width="50" height="50" usercard=
"id=1649763572&usercardkey=weibo_mp&refer_flag=1001030103_" class="W_face_radius"></a>
</div>
<div class="content clearfix" node-type="like">
<div class="feed_content wbcon">
        <a class="W_texta W_fb" nick-name="御遠" href="http://weibo.com/lyle?refer_flag=1001030103_" target="_blank" title="御遠" usercard=
        "id=1649763572&usercardkey=weibo_mp&refer_flag=1001030103_"      suda-data="key=tblog_search_weibo&value=weibo_ss_4_name" class=
        "name_txt W_fb">
    御遠        <a target="_blank" href="http://verified.weibo.com/verify" title= "微博个人认证 " alt="微博个人认证 " class="W_icon
    icon_approve"></a><a href="http://vip.weibo.com/personal?from=search" target="_blank" title="微博会员"><i class="W_icon ico_member6"
    ></i></a></a>
            <p class="comment_txt" node-type="feed_list_content" nick-name="御遠">
    <a class="a_topic W_linkb" href="http://huati.weibo.com/k/%E7%8E%8B%E8%80%85%E8%8D%A3%E8%80%80?from=526"suda-data=
    "key=tblog_search_weibo&value=weibo_feed_topic" target="_blank">#<em class="red">王者荣耀</em>#</a> 真 · 一打五…
    对面李白被我A死之后就再没敢来抓过我… <img src="http://img.t.sinajs.cn/t4/appstyle/expression/ext/normal/2c/moren_yunbei_org.png" title=
    "[允悲]" alt="[允悲]" type="face" />
    </p>

<div class="WB_media_wrap clearfix" node-type="feed_list_media_prev">
          <!-- 图片多余一张只显示音频视频的smallcard -->
            <!-- 九宫格 -->
  <div class="media_box" node-type="fl_pic_list" action-data=
  "uid=1649763572&pic_ids=625564f4gy1fkzfw0jy52j20qo1bftfj,625564f4gy1fkzfw01w3bj20qo1bfn31&mid=4168301607060712">
   <ul class="WB_media_a WB_media_a_mn clearfix">
      <li class="WB_pic S_bg2 bigcursor">
   <img class="bigcursor" src="http://ww1.sinaimg.cn/square/625564f4gy1fkzfw0jy52j20qo1bftfj.jpg" action-data=
   "pic_id=625564f4gy1fkzfw0jy52j20qo1bftfj" action-type="fl_pics"          suda-data="key=tblog_search_weibo&value=weibo_ss_4_pic"/>
  </li>
      <li class="WB_pic S_bg2 bigcursor">
   <img class="bigcursor" src="http://ww3.sinaimg.cn/square/625564f4gy1fkzfw01w3bj20qo1bfn31.jpg" action-data=
   "pic_id=625564f4gy1fkzfw01w3bj20qo1bfn31" action-type="fl_pics"          suda-data="key=tblog_search_weibo&value=weibo_ss_4_pic"/>
  </li>
```

图 16-17 相关微博内容

每一条微博信息都存放在 <div class = "content clearfix" node-type = "like"> 中，在此标签内，可以分别找出微博用户、发布内容、发布图片和发布视频所在位置。

（1）微博用户在 <a> 标签，属性 class="W_texta W_fb"。

（2）发布内容在 <p> 标签，属性 class="comment_txt"。但有一种特殊情况是，文字内容过长的时候，需要单击“展开全文”才能看到完整的内容，而“展开全文”存放在 <p> 里面的 <a> 标签 href=”javascript:void(0);”中，在浏览器中单击“展开全文”触发的是一个 Ajax 请求，如果获取全文内容，就需要爬取 Ajax 请求。

（3）发布的图片在 <img> 标签，属性是 class="bigcursor"。

（4）发布的视频在 <a> 标签，属性 class 含有 "WB_video"。

已经确定采集数据的具体位置，实现代码如下：

```
from bs4 import BeautifulSoup
import re
import urllib
import csv
import requests
import time
import datetime
from concurrent.futures import ThreadPoolExecutor

# 多线程爬取视频文件
def thread_video(get_video_value, video_path):
    if get_video_value:
        url = str(get_video_value).split('video_src=')[
            1].split('cover_img=')[0][0:-1]
        url = urllib.parse.unquote(url) + '=' + str(int(time.
time() * 1000))
        if 'http:' not in url:
            url = 'http:' + url
        try:
            temp_value = requests.get(url)
            video = open('video/' + video_path, 'wb')
            video.write(temp_value.content)
            video.close()
        except BaseException:
            pass

# 多线程爬取图片
def thread_img(k, img_path):
    if 'http:' in k['src']:
        img_r = requests.get(k['src'])
    else:
        img_r = requests.get( 'http:'+ k['src'])
```

```
        img = open('image/' + img_path, 'wb')
        img.write(img_r.content)
        img.close()

    # 采集微博
    def collect(keyword, session, pagenumber=1):
        # 关键字编码
        keyword_change = urllib.parse.quote_plus(keyword)
        keyword_change = urllib.parse.quote_plus(keyword_change)
        now = datetime.datetime.now().strftime('%Y-%m-%d')
        # 构建 URL，地区默认广东省广州市
        url = 'http://s.weibo.com/weibo/' + keyword_change + '&region
=custom:44:1&typeall=1&suball=1&page=%s' % (str(pagenumber))
        r = session.get(url)
        # 解决中文乱码
        get_value = r.content.decode('unicode_escape').replace('\/', '/')
        # 清洗多余数据
        index = get_value.find('<div class="face">')
        get_value = get_value[index::]
        soup = BeautifulSoup(get_value, 'html5lib')
        # 获取当页全部用户信息
        get_info = soup.find_all('div', re.compile('content clearfix'))

        for i in get_info:
            # 获取用户信息
            get_user = i.find('a', re.compile('W_texta W_fb'))
            user_name = get_user.getText().replace('\n', '').strip()
            # 获取文字全部内容
            get_comment = i.find('p', re.compile('comment_txt'))
            # 文字过长需要特殊处理
            get_long_comment = get_comment.find('a', href=re.
compile('javascript'))
            if get_long_comment:
                get_url = 'http://s.weibo.com/ajax/direct/
morethan140?'+ get_long_comment['action-data'] + '&_t=0&__rnd=' +
str(int(time.time() * 1000))
                temp_value = session.get(get_url)
```

```
            get_comment_temp = temp_value.json()
            get_comment = get_comment_temp['data']['html']
            get_comment = BeautifulSoup(get_comment, 'html5lib')
        # 输出全部文字内容
        comment = get_comment.getText().replace('\n', '').strip()
        comment = comment.encode("utf-8", 'ignore').
decode('UTF-8', 'ignore')

        # 获取图片
        img_path_list = ''
        get_img_value = i.find_all('img', re.compile('bigcursor'))
        # 输出多张图片
        for k in get_img_value:
            img_path = str(int(time.time() * 1000)) + '.jpg'
            img_path_list = img_path_list + img_path + '/'
            pool = ThreadPoolExecutor(max_workers=1)
            pool.submit(thread_img, k, img_path)

        # 输出视频
        video_path = ''
        get_video_value = i.find('a', re.compile('WB_video'))
        if get_video_value:
            pool = ThreadPoolExecutor(max_workers=1)
            video_path = str(int(time.time() * 1000)) + '.mp4'
            pool.submit(thread_video, get_video_value, video_
path)

        # 生成 csv
        f = open('data.csv', 'a', newline='', encoding='gb18030')
        writer = csv.writer(f)
        writer.writerow([user_name, comment, img_path_list,
video_path, now])
        f.close()
```

整段代码共由以下三个函数组成。

- thread_img()：多线程下载图片。
- thread_video()：多线程下载视频。

- collect()：实现微博采集，函数参数 keyword、session 和 pagenumber 分别是关键字、带有用户信息的会话对象和采集页数。因为本节的代码存放在文件 weibo_collect.py 中，与用户登录的代码不在同一个文件，所以需要将带有用户信息的会话对象传递给该函数。

函数 collect() 实现的功能依次如下：

（1）对函数参数 keyword 执行两次 URL 编码。

（2）构建请求链接并发送请求，对获取请求后的响应内容进行编码处理。

（3）对响应内容进行清洗处理，采集微博用户信息、文字内容、图片和视频。文字过长时，需要向网站发送请求获取完整的文字内容；图片和视频分别调用函数 thread_img() 和 thread_video()，使用多线程下载文件，提高爬取速度。

代码运行需要结合 16.3 节的登录功能一起使用，将本节代码存放在文件 weibo_collect.py 中，与文件 weibo_login.py 同一目录，当前目录下必须有文件夹 image 和 video，否则下载的图片和视频无法保存。打开修改微博登录文件 weibo_login.py，代码如下：

```
if __name__ == "__main__":
    # 构造请求头
    agent = 'Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0)
Gecko/20100101 Firefox/41.0'
    headers = {
        'User-Agent': agent
    }
    # 代理 IP
    proxies = {}
    # 新建会话
    session = requests.session()
    # 第三方平台账号、密码
    yundama_username = 'xxxx'
    yundama_password = 'xxxx'
    user_info = login('13435423143','xxxx')
    # 导入微博采集功能
    from weibo_collect import collect
```

```
# 爬取前 10 页数据
for i in range(10):
    collect('# 王者荣耀 #',session,i)
```

16.5 发布微博

发布微博是在浏览器上编辑好要发布的内容，然后单击“发布”按钮进行发布。微博中有很多可以编辑的功能，如插入表情、话题、图片、视频和定时发送等。其中，表情和话题可以归纳为文字内容。本节主要实现文字内容、图片和定时发送的微博发布。

在浏览器上分别捕捉三种不同的发布方式的请求，如图 16-18~ 图 16-20 所示。

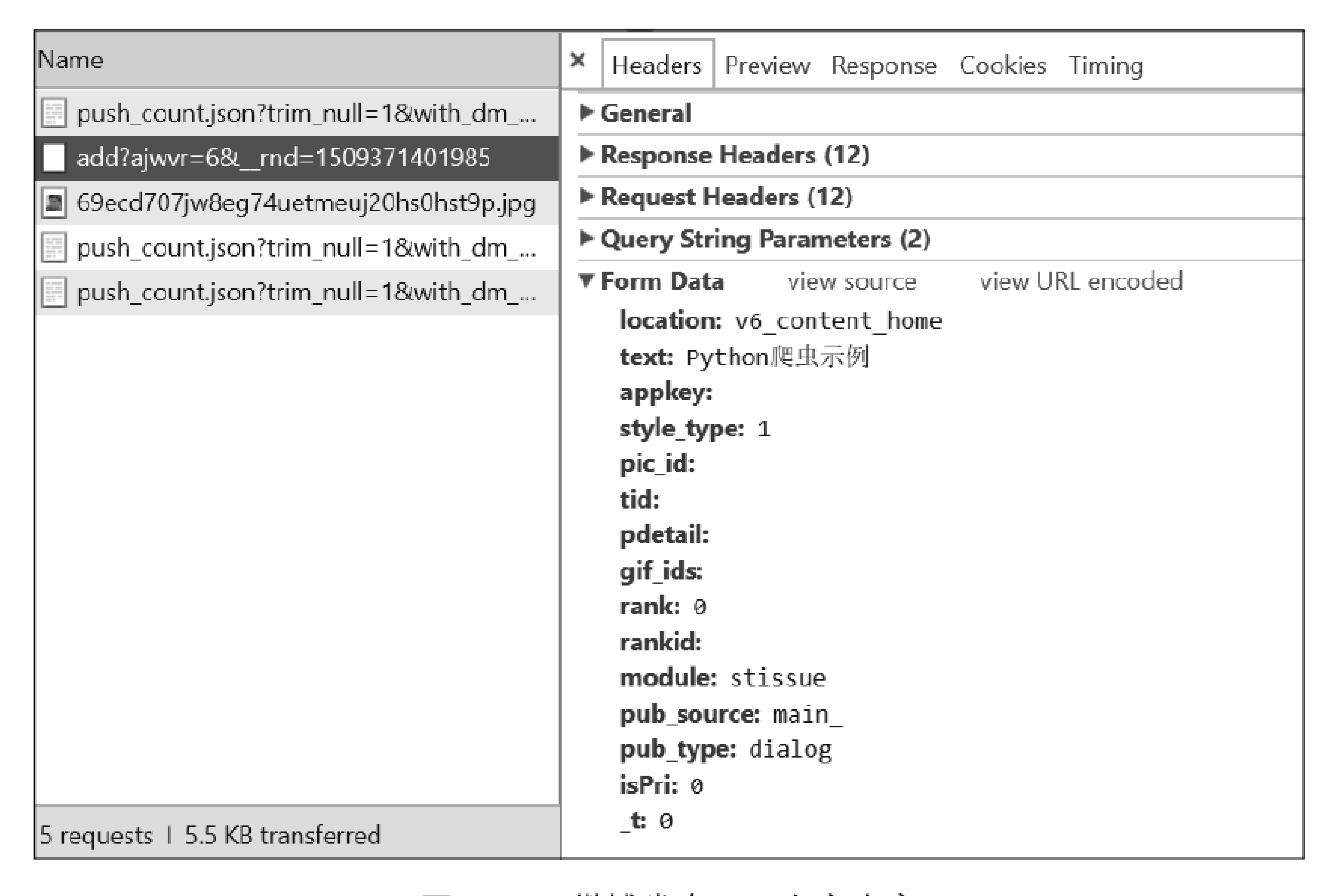


图 16-18　微博发布——文字内容

通过对比三种不同的发布方式的请求可以发现，三者的请求链接一致，唯一区别在于请求参数的差异。请求参数的差异如下：

（1）对比图 16-18 和图 16-19 的请求参数，图 16-19 多出参数 updata_img_num，该参数是所发布的图片的数量。参数 pic_id 的值非空，从参数名分析，参数 pic_id 应该是图片的 id。

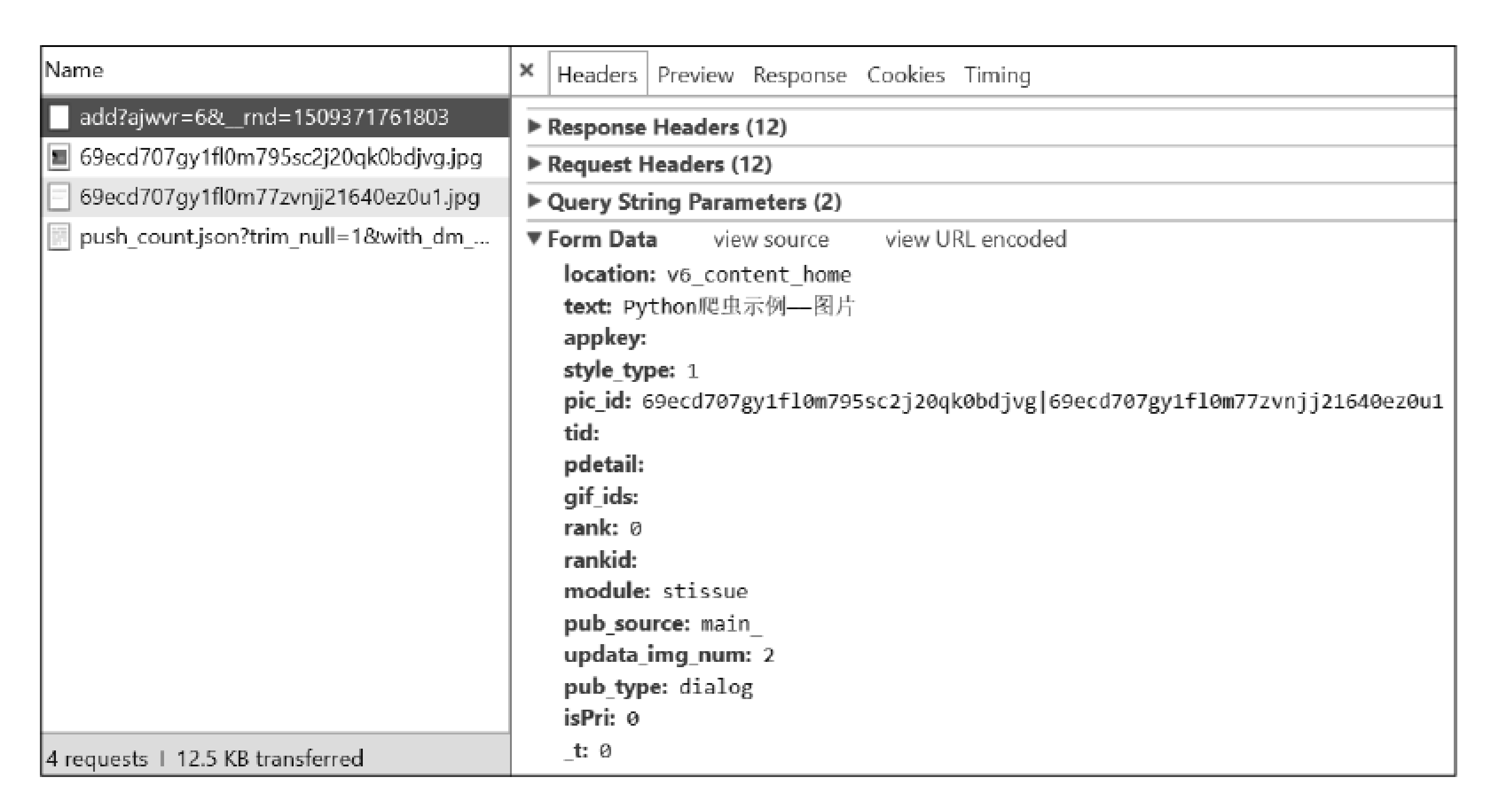


图 16-19 微博发布——文字内容和图片

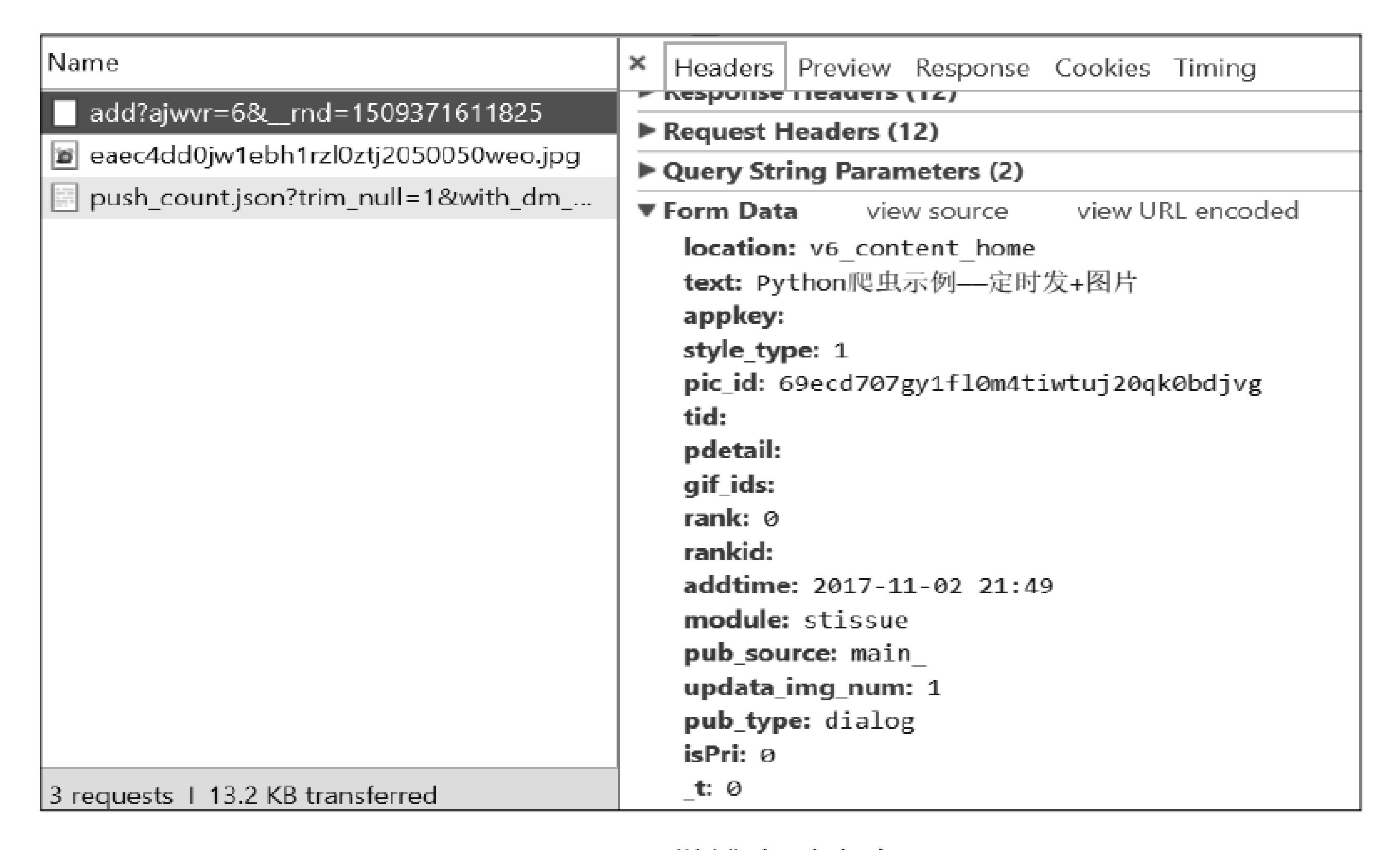


图 16-20 微博定时发布

（2）对比图 16-18 和图 16-20 的请求参数，图 16-20 多出参数 addtime，该参数是发布时间；再对比两者的参数 pic_id 和 updata_img_num，图 16-19 比图 16-20 多一张图片，图 16-19 的参数 pic_id 将每张图片之间用“|”隔开。

（3）参数 location 和 text 分别是用户信息和发布的文字内容，其他参数都是固定不变的。

经过上述分析，现在无法确定 pic_id 的数据来源，该参数如果是图片的 id，那么在添加图片的时候，网站应该会对添加的图片生成一个图片 id，用于标识图片。为了验证猜想，我们捕捉添加图片时所触发的请求信息，如图 16-21 所示。

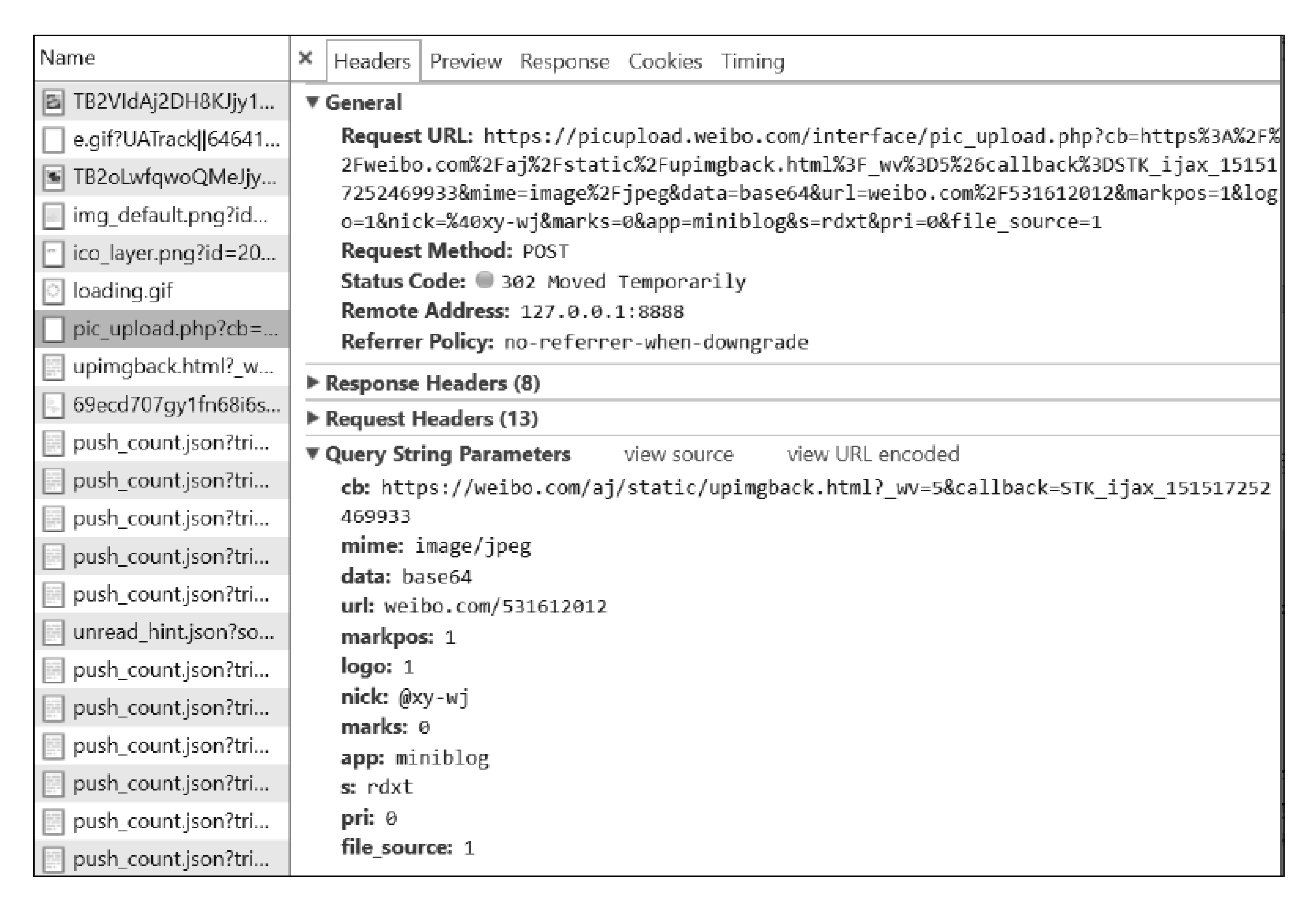


图 16-21 图片添加信息

从图 16-21 的请求信息分析，请求链接是 GET 请求，请求方法是 POST 请求，而且请求参数已在请求链接上，说明该请求 POST 的数据不是请求参数，而是 POST 图片文件，为了进一步验证猜想，我们使用 Fiddler 分析该请求信息，如图 16-22 所示。

```
POST https://picupload.weibo.com/interface/pic_upload.php?cb=https%3A%2F%2Fweibo.com%2Faj%2Fstatic%2Fupimgback.html%3F_wv%3D5%
Host: picupload.weibo.com
Connection: keep-alive
Content-Length: 368765
Cache-Control: max-age=0
Origin: https://weibo.com
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3218.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Referer: https://weibo.com/531612012/home
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Cookie: SINAGLOBAL=6464114471908.122.1513049031838; login_sid_t=bb7b65365f5c70699b9da61abf66bca5; cross_origin_proto=SSL; _s_t
```

b64_data=iVBORw0KGgoAAAANSUhEUgAAAu4AAAd2CAYAAABSEChDAAAACXBIWXMAAAsTAAALEwEAmpwYAAAKTWlDQ1BQaG90b3Nob3AgSUNDIHByb2ZpbGUAAHjan
2FblqWfW8tQz6yKf0d9Z5DdQxZWpsJRNB2RVAxJ1ETGa3mptYGhZUFPIx%2BWR2FSounRxVudoiihg2TZVpKtnAKAP4ToRl0dRzy30C1XLPERDlef38iaBTMzErnqa
2FDjcUyzAKfkPNFVaXC4IfjNfShY5eGDE8xdH3UQogGhloq0IMMATqdJgtrALUFqKVNOIF55B1VBaLKMzGeKYq3nWDZWMMIRPWuKLemINhRwWclD1aZ%2BBNqJBnrD
2ByzZjSaI7OOFwWqSbpc5rO5z9QxyguNuMZoC71CKacbkYw9VnJjYb9WJNFIGAXQsNqy5MNozyvAWpCV2z9%2BTO7nSBL42CGz4XFW4vXDT29u3QTtFGFDuA0KQgOk
2BaqefuZJA9wfe%2FQ18nu%2F8w%2BoOREUubOS7%2Fyu75ff810%2FkPzbVZDF%2FtZcnoQFtWe8BfTu%2Fa4ixwni%2FaDV%2FVOTNUuqp86A%2BysJuG9ZnEbjc
2F6hHIaK5J2mbLRj6zOccNjcUly7HbfHavXxIVk6jlElpfqseTPVeQnrewKnqbAAJqIVtXqWR8U6onc9FS6%2BrO%2Fj3Ihc8xORLANU21K2kwKADQAMHOHyphb27F
2BzB23Ti7xS6yxOb8NaYnbJIsuNT4RVciPZ5QJt3emYNiRtIrIuzTeNLyOZpKVMnVCdiUDa6POKrJm%2B4M2XRL5p2wa8zg1YAn4psQ5aacvhlUkyUrh7czc5DmEMw
2BJVaP9PZdpQ1bCxPnaOpouenITpHxUFRPVcS4kjqBnuqHa2pnqGcFKONoYmBTiC7aL8jwwcZlH755d1bkQ49K5XC2Mrf2gotqv1DXFVJ%2Fn7ly26SCxI49pyNr2N
2FQGUKwqOIra9PrJtPrJyvsjVgHCV78wZYdBmIA6dWLxIbFo1Erdt9doi8OCOYph1EcmiioWyyody2xOk10gk3kuru9QDR3L6ORJVXuQEnxrsG%2BcPoYRXjIs6s%2
2BfPXX389rVy5MhHjt9xyC1166aUzfiOA4OSBRURwGAAAAAAAAFjYIMcdAAAAAAAACHcAAAAAAAAAhDsAAAAAAAAQ7gAAAAAAAAAIdwAAAAAAAACEOwAAAAAAABDuA
2B9dZbaT87ceKERkdHGSQAeIBQ4w4sk%2B3bt2vPnj2yWqOpl2%2FatEn%2F%2Fb%2F%2Fd%2F3N3%2FxNWigrJMDdvXtXg4ODcrlc6uzslMViKflruHz5sgKBgJ58
2F8Iq%2Fr79%2B%2Fv6i69%2FlKtRGSIZFIaHh4WIFAQO3t7XI4HIrFYmboTp793rJli4aHh%2FXJJ5%2BoO6Mj430Zdedut1sul0sbN240x2o%2Bm82m3bt3mzPgd
2BNyVql97W1tb1rKDQmQKfUbZSqnLhH73u9%2BZvyvzu4I4nU5VV1frnXfeSTvbcfHiRVmtVm3ZsiXr75IRGo2f19TUaOfOnXI4HGpra1MgENDVq1cVCATM64yOjub
2FqWss%2FPCHP6z4PgHA7XbLz6Ge3Lp1q%2BgzVrMhhFC4E0LIAkcb75xKpRAKhVTCvZRoHh0d1RXu9UywPHfuHK5evVrzdW7evFn2mHrXkc%2FlchVfr5HhM6lUqi
2B594Pb7S65P4d2rNQasgvFIUAIhTt5JNi4cSOampoavuyuHKirrSozV1gsFmzfvhOXLlwwFeddKvRBL6zAbPJZLXR3d%2BPSpUvo6Oioeina4XDIIUaFQmHeRftc9
2BfkSjUbz%2F%2Fvv43d%2F93aq8snOFGS9epWzfvh1XrlzR%2FdvixYuxceNGlSi4ceMGAGDZsmWmvOOSJOTzecOVgWg0inQ6rRKN4XAYoVCoZjEYi8Xq0k7btm1D
2Frs1q1bRd61QCCgOkYvVMescFHGL9dSOanaGPdcLgebzYZCoaDy7Jmtga2MCRdtqOdhFHtEVEq92qdSg12Z3yDya5Qx79V6%2FiOWiyosRYvdblcZmOPj46q%2FL1

图 16-22 图片添加信息

从图 16-22 看到，图片上传 POST 的数据是 b64_data，该数据是使用 base64 对图片的字节流加密而成的。综合上述分析，实现微博发布功能需要实现有两部分功能：第一是实现图片上传，获取图片 id；第二是根据条件判断选择微博发布方式。代码如下：

```
    import base64
    import time
    # 获取上传图片 id
    def upload_pic(session, watermark, nick, file_list=[]):
        pic_id_list = []
        # 判断图片数量是否在 1 ~ 9 之间
        if len(file_list)>0 and len(file_list)<10:
            for i in file_list:
                url = 'https://picupload.weibo.com/interface/pic_
upload.php?cb=https%3A%2F
                    %2Fweibo.com%2Faj%2Fstatic%2Fupimgback.html%3F_
wv%3D5%26callback%3DSTK_ijax_'+
                    str(int(time.time()*100000))+'&mime=image%2Fjpeg&
data=base64&url=weibo.com%2F'+
                    watermark+'&markpos=1&logo=1&nick=%40'+nick+'&mar
ks=0&app=miniblog'
                # 图片以字节数据流读取，然后以 base64 加密
                files={'b64_data':base64.b64encode(open(i, "rb").
read())}
                # 上传文件
                r = session.post(url, files=files)
                print(r.text)
                # 获取图片 id
                get_picid=eval(r.text.split('</script>')[1])['data']
['pics']['pic_1']['pid']
                pic_id_list.append(get_picid)
        return pic_id_list

    # 发送微博，pic_id_list 是上传图片 Id 列表
    def send(session,watermark, location, value, addtime='',pic_id_
list=[]):
        # 构建请求头
        headers = {'Referer':'http://weibo.com/'+str(watermark)+'/
home',
        'user-agent':'Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0)
Gecko/20100101 Firefox/41.0'}
        # 构建请求参数
```

```
        data = {'location': location, 'text': value, 'appkey':
                '', 'style_type': '1', 'pic_id': '','tid': '',
                'pdetail': '','gif_ids':'','addtime':addtime,
                 'rank': "0", 'rankid': '',
                      'module': 'stissue','pub_type': 'dialog', 'pub_
source': 'main_', '_t': '0'}
        # 发送图片
        if pic_id_list:
            pic_id=''
            for i in pic_id_list:
                pic_id += i+'|'
            # 去除最后的"|"
            if pic_id[-1]=='|':
                pic_id=pic_id[0:len(pic_id)-1]
            data['updata_img_num'] = str(len(pic_id_list))
            data['pic_id'] = pic_id
        # 构建 URL
        url='https://www.weibo.com/aj/mblog/add?ajwvr=6&__rnd=%s'
%(int(time.time()*1000))
        r = session.post(url, data=data, headers=headers)
        if r.status_code==200:
            return True
        else:
            return False
```

上述是本节实现的功能代码，存放在文件 weibo_send.py 中，整段代码由以下两个函数组成。

- upload_pic()：实现图片上传。函数参数 session、watermark、nick 和 file_list：
  - session 是带有用户登录状态的会话对象。
  - watermark 和 nick 是用户信息。
  - file_list 是图片列表，列表元素是图片路径。
- send()：实现微博发布。函数参数有 session、watermark、location、value、addtime 和 pic_id_list：

- session 是带有用户登录状态的会话对象。
- watermark 和 location 是用户信息。
- value 和 addtime 是发布内容和发布时间。
- pic_id_list 是函数 upload_pic() 返回的图片 id 列表。

send() 功能说明如下：

（1）需要重新设置请求头并加入 Referer 信息，否则会导致发送失败，因为网站做了检测 Referer 的反爬虫机制。

（2）请求参数合并了三种不同的发布方式，例如只发布文字内容，只需将参数 pic_id 和 addtime 的值设置为空即可。若发布图片，在设置 pic_id 的参数值时，则会相应地创建参数 updata_img_num。

（3）请求链接最后的一串数字是当前时间的时间戳再乘以 1000 后取整所得。

代码与 16.4 节的运行方式一样，打开修改微博登录文件 weibo_login.py，代码如下：

```
if __name__ == "__main__":
    # 构造请求头
    agent = 'Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0) Gecko/20100101 Firefox/41.0'
    headers = {
        'User-Agent': agent
    }
    # 代理 IP
    proxies = {}
    # 新建会话
    session = requests.session()
    # 第三方平台账号、密码
    yundama_username = 'xxxx'
    yundama_password = 'xxxx'
    user_info = login('13435423143','xxxx')
    # 导入微博发布模块
    from weibo_send import upload_pic,send
    # 获取用户信息
    watermark = user_info['watermark']
```

```
        nick = user_info['nick']
        location = user_info['location']
        # 设置图片列表
        file_list=['aa.png','bb.png']
        # 获取图片 id 列表
        pic_id_list = upload_pic(session,watermark,nick,file_list)
        # 发布微博
        send(session, watermark, location, "Python 爬虫 ", addtime='',
pic_id_list=pic_id_list)
```

16.6 关注用户

在微博中关注用户有两种方式：

（1）在用户的某条微博上，在将鼠标移到用户头像时所弹出的窗口中单击“关注”。

（2）在微博用户的首页单击“关注”。

两种关注方式的请求链接是一样的，区别在于请求参数的差异。本项目主要实现第二种关注方式，在浏览器上捕捉其请求信息，如图 16-23 和图 16-24 所示。

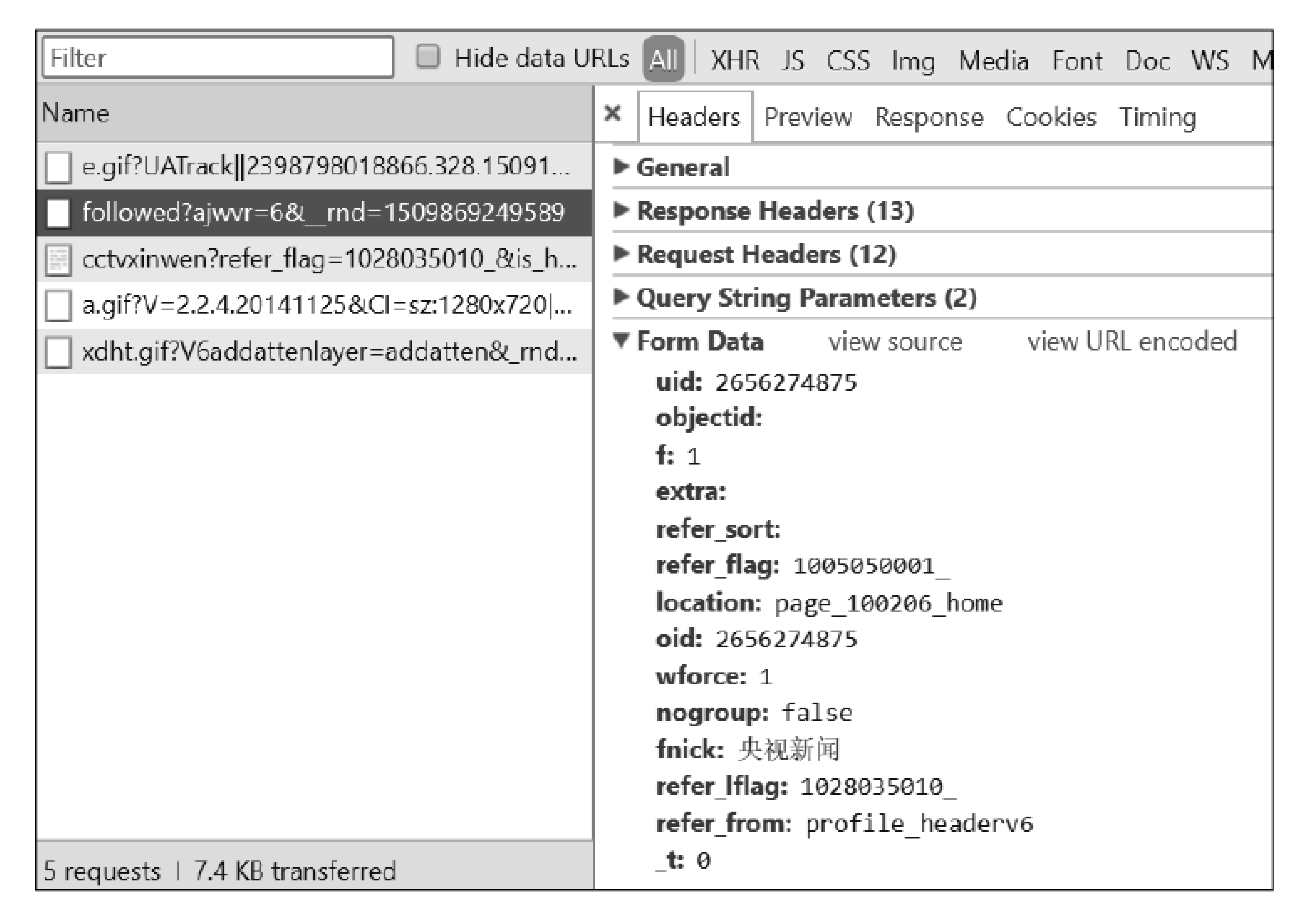


图 16-23 关注用户时的请求信息



图 16-24 关注用户时的响应内容

从图 16-23 的请求参数分析可得，参数 uid、location、oid 和 fnick 是被关注用户的信息，暂时无法得知被关注用户信息的来源。其余的参数是固定不变的。

从图 16-24 的响应内容分析可得，用户被关注成功之后，网站主要返回 JSON 数据。观察数据内容，可从“code”的值来判断是否关注成功。

进一步核实被关注用户信息的来源，以“央视新闻”的微博为例，分析查找浏览器在“央视新闻”的微博首页所捕捉的请求信息，最终在 Doc 标签找到该微博信息，如图 16-25 所示。

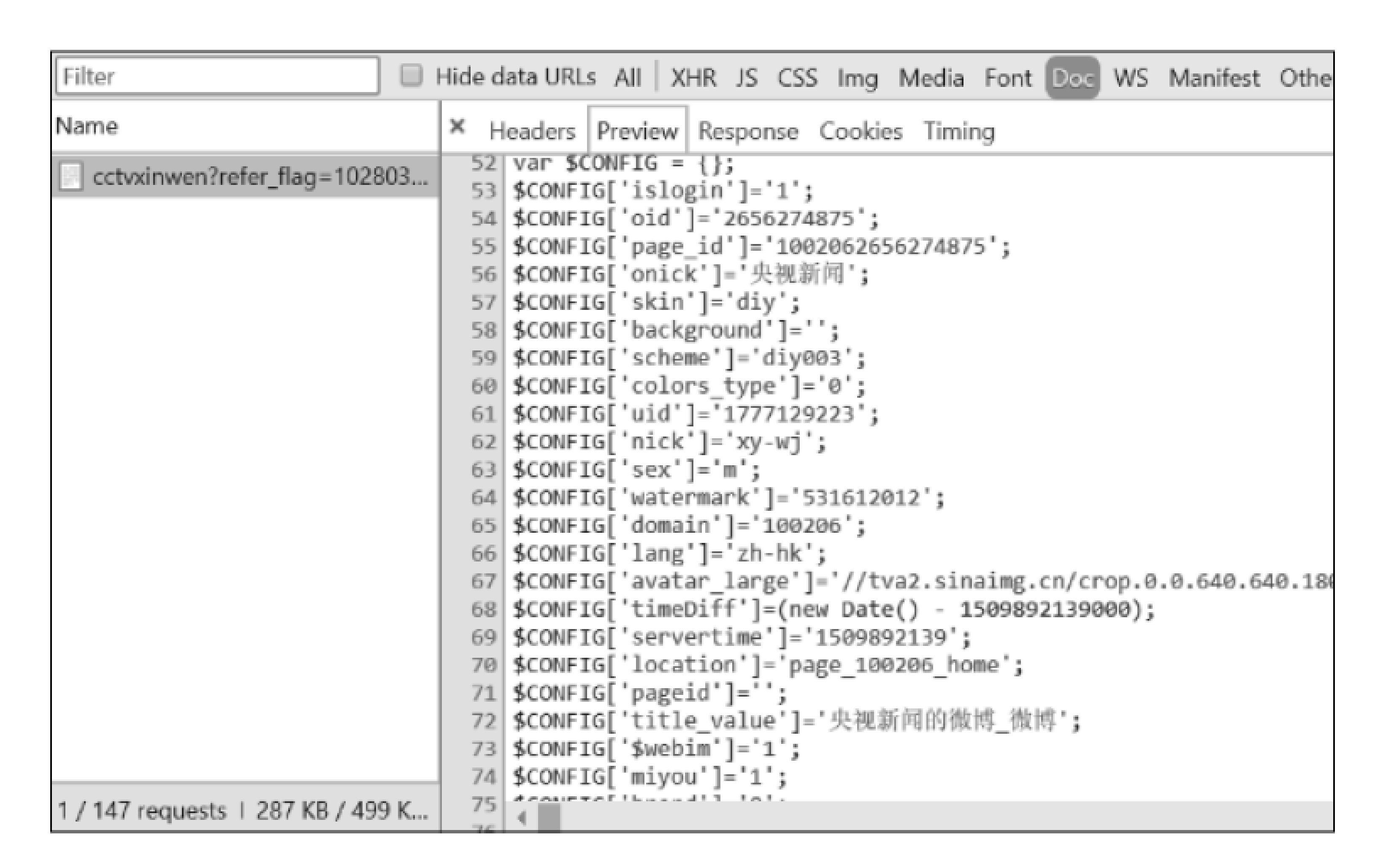


图 16-25 被关注用户的首页信息

从图 16-23 和图 16-25 的数据对比得出，图 16-23 的参数 uid、location、oid 和 fnick 分别对应图 16-25 的 oid、location、oid 和 onick。

综合上述分析，实现代码如下：

```
import time
# 关注微博，session 是用户登录后的会话，follow_url 是关注用户的微博主页
def follow_weibo(session, follow_url):
    # 构建请求头
    agent = 'Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0) 
Gecko/20100101 Firefox/41.0'
    headers = {
        'User-Agent': agent,
        'Referer': follow_url
    }
    follow_info = {'oid': '', 'onick': '', 'location': ''}
    r = session.get(follow_url)
    response = r.text
    follow_info['oid'] = response.split("$CONFIG['oid']='")[1].
split("';")[0]
    follow_info['onick'] = response.split("$CONFIG['onick']='")
[1].split("';")[0]
    follow_info['location'] = response.
split("$CONFIG['location']='")[1].split("';")[0]
    # 关注 URL，参数__rnd 为时间戳
    url = 'https://www.weibo.com/aj/f/followed?ajwvr=6&__rnd=' + 
str(int(time.time() * 1000))
    data = {
        'uid': follow_info['oid'],
        'objectid': '',
        'f': '1',
        'extra': '',
        'refer_sort': '',
        'refer_flag': '1005050001_',
        'location': follow_info['location'],
        'oid': follow_info['oid'],
        'wforce': '1',
        'nogroup': 'false',
```

```
            'fnick': follow_info['onick'],
            'refer_lflag': '',
            'refer_from': 'profile_headerv6',
            '_t': '0'
        }
        r = session.post(url, data=data, headers=headers)
        # 判断是否关注成功
        if (r.json()['code']) == '100000':
            return (follow_info['onick'] + ' 关注成功 ')
        else:
            return (follow_info['onick'] + ' 关注失败 ')
```

上述是本节实现的功能代码，存放在文件 weibo_follow.py 中，代码说明如下：

（1）函数 follow_weibo 的参数分别是带有用户信息的会话对象和被关注的微博首页链接。

（2）重新构建请求头，主要在发送关注用户的请求时所使用。

（3）访问被关注用户的首页，获取被关注用户的信息。

（4）对获取的数据构建请求参数，实现发送用户关注请求。

代码与 16.5 节的运行方式一样，打开修改微博登录文件 weibo_login.py，代码如下：

```
if __name__ == "__main__":
    # 构造请求头
    agent = 'Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0)
Gecko/20100101 Firefox/41.0'
    headers = {
        'User-Agent': agent
    }
    # 代理 IP
    proxies = {}
    # 新建会话
    session = requests.session()
    # 第三方平台账号、密码
    yundama_username = 'xxxx'
    yundama_password = 'xxxx'
```

```
user_info = login('13435423143','xxxx')
# 导入关注用户模块
from weibo_follow import follow_weibo
# 关注用户的首页链接
follow_url = 'https://weibo.com/renminwang'
status = follow_weibo(session, follow_url)
print(status)
```

16.7 点赞和转发评论

本节主要实现两个功能：点赞和转发评论。两者实现方式和 16.6 节的实现方式大致相同，主要在对方的微博首页实现。

在浏览器上访问某微博的用户首页，以“有妖气原创漫画梦工厂”为例（https://weibo.com/u17t），在该微博用户的第一条微博中单击“点赞”按钮，开发者工具所捕捉的请求信息如图 16-26 所示。

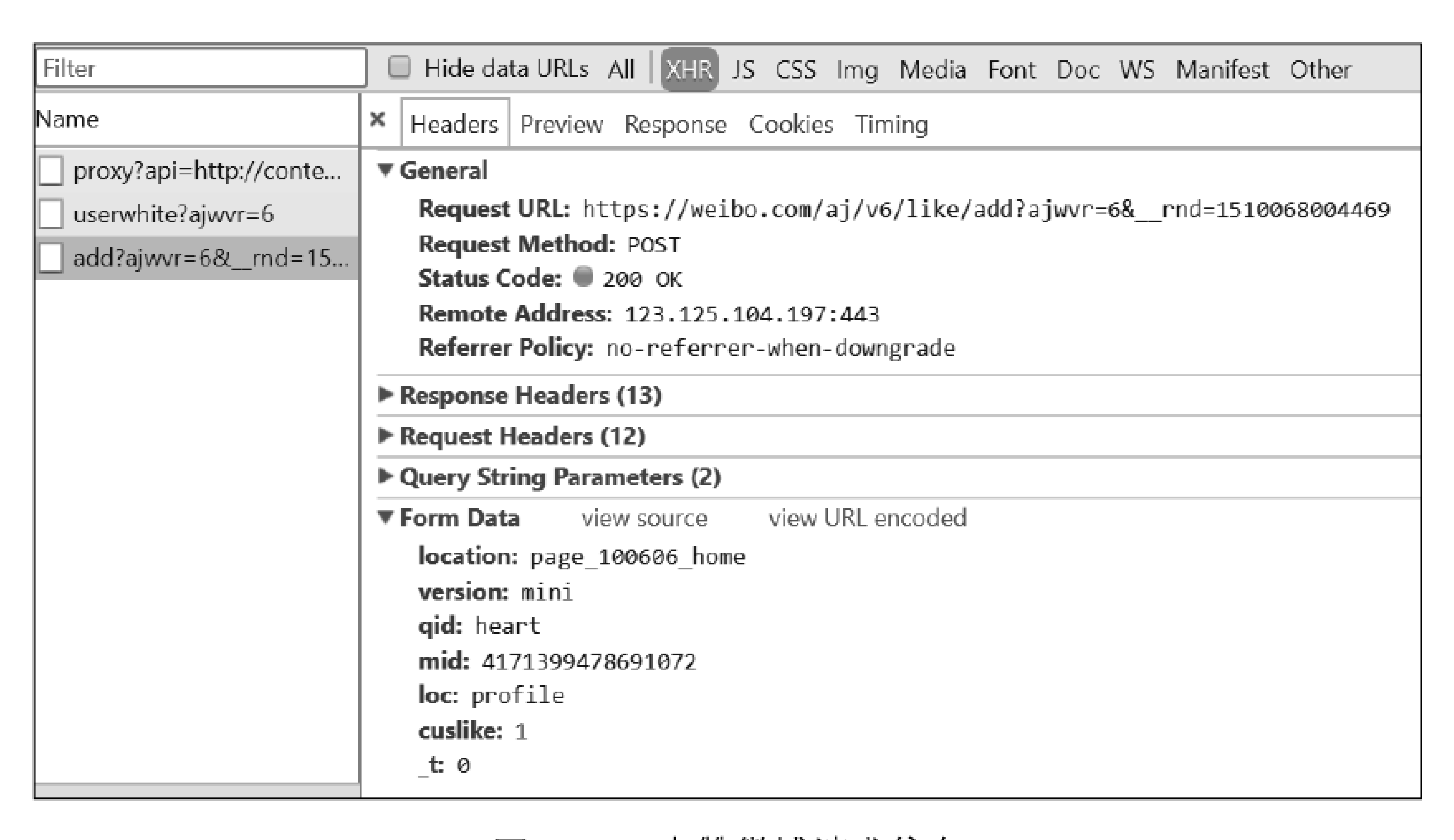


图 16-26　点赞微博请求信息

从图 16-26 的请求分析可知，请求链接的 _rnd 是当前时间的时间戳乘以 1000 再取整所得，请求参数分析如下：

（1）参数 location 代表被点赞的微博用户信息，数据来源可在 Doc 标签返回的 HTML 内容中找到。

（2）参数 mid 数据来源无法得知，点赞不同的微博，其参数值随之变化。

（3）其余参数值固定不变。

根据参数 mid 的变化规律得知，不同微博的数据会随之变化，那么参数 mid 可能用于标识微博的唯一性。为了验证猜想是否正确，我们分析浏览器所捕捉到的请求信息，最终在 Doc 下找到 mid 的参数值，如图 16-27 所示。



图 16-27 查找请求参数

在 Doc 标签返回的 HTML 内容中快速查找（Ctrl+F）参数值（4171399478691072），在 HTML 里的 JavaScript 代码中找到参数 mid，而且参数 mid 是重复出现的。因此可以确定，参数 mid 可在网站返回的 HTML 中找到。综合上述分析，实现代码如下：

```
import re
import time
# session 是会话对象，like_url 是用户首页
def like_weibo(session, like_url):
    # 获取点赞用户的前 16 条微博
    r = session.get(like_url)
    # 获取 location
    location = r.text.split("$CONFIG['location']='")[1].
split("';")[0]
    # 获取 mid
    mid_list = re.findall(r'mid=(.\d+)&name', r.text, re.S)
```

```
        # 构建请求头
        agent = 'Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0)
Gecko/20100101 Firefox/41.0'
        headers = {
            'User-Agent': agent,
            'Referer': like_url
        }
        # 点赞功能，默认点赞第一条微博
        url = 'https://weibo.com/aj/v6/like/add?ajwvr=6&__rnd=' +
(str(int(time.time() * 1000)))
        data = {
            'location': location,
            'version': 'mini',
            'qid': 'heart',
            'mid': mid_list[0],
            'loc': 'profile',
            'cuslike': '1',
            '_t': '0'
        }
        r = session.post(url, data=data, headers=headers)
        # 根据返回内容判断是否成功
        if (r.json()['code']) == '100000':
            return ('点赞成功')
        else:
            return ('点赞失败')
```

点赞功能定义为函数 like_weibo()：函数参数 session 是会话对象；like_url 是被点赞用户的首页链接。函数实现的功能如下：

（1）访问被点赞用户的首页链接，获取用户的 location 信息和 mid_list。mid_list 是当前用户的前 16 条微博 mid 组成的列表。

（2）构建请求头，作为发送点赞请求的请求头。如果不加请求头，该请求就会被服务器视为非法请求，因为服务器会对请求头的 Referer 进行检查，这是一种反爬虫机制。

（3）发送点赞请求，将获取的 location 和 mid_list 作为请求参数，mid_list 默认取第一位元素，即默认点赞第一条微博。

（4）针对请求后的响应内容判断是否点赞成功。

完成微博点赞功能后，接着完成转发评论功能，该功能的实现方式和点赞功能类似。以上述微博用户首页为例，单击转发该用户的第一条微博，勾选“同时评论”选项，在开发者工具看到该请求信息，如图 16-28 所示。

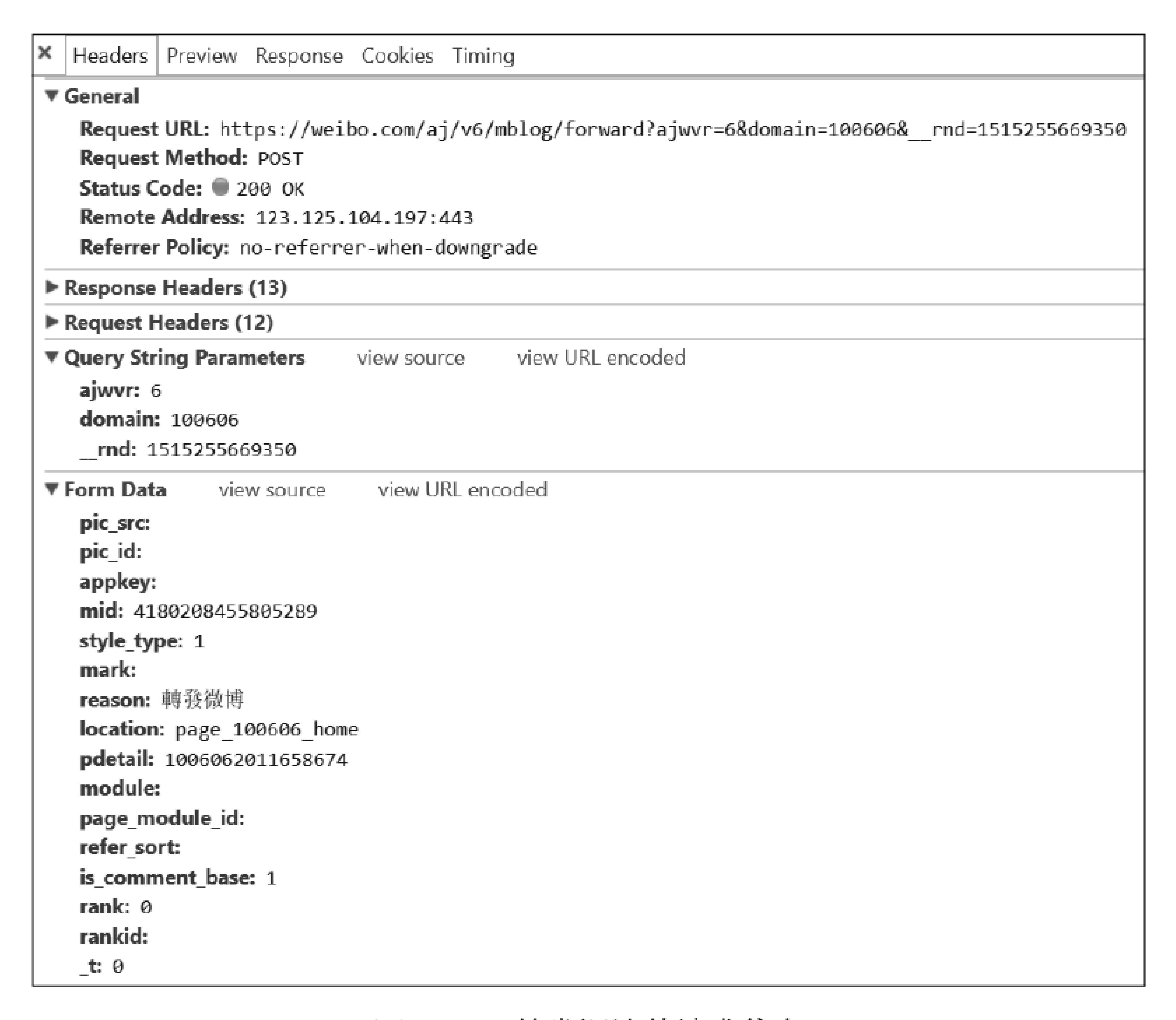


图 16-28 转发评论的请求信息

从请求信息分析可得：请求链接的 _rnd 是当前时间的时间戳乘以 1000 再取整所得，domain 是被转发的用户信息，请求参数分析如下：

（1）参数 location 和 mid 与点赞功能的请求参数一致。

（2）参数 reason 是转发内容。

（3）参数 pdetail 无法确定。

（4）参数 pic_id 与 16.5 节的请求参数 pic_id 一致。

（5）参数 is_comment_base 代表转发时的“同时评论”选项。

（6）其余参数值固定不变。

为了确定参数 pdetail 的数据来源，查找分析浏览器所捕捉到的请求信息，最后在 Doc 下找到该参数的数据来源，该参数代表被转发微博的用户信息，如图 16-29 所示。

图 16-29　请求参数信息

综合上述分析，实现代码如下：

```
# 转发评论微博
def forward_weibo(session, forward_url, reason):
    # 获取点赞用户的前 16 条微博
    r = session.get(forward_url)
    # 获取 location
    location = r.text.split("$CONFIG['location']='")[1].
split("';")[0]
    page_id = r.text.split("$CONFIG['page_id']='")[1].split("';")[0]
    domain = r.text.split("$CONFIG['domain']='")[1].split("';")[0]
    # 获取 mid
    mid_list = re.findall(r'mid=(.\d+)&name', r.text, re.S)
    # 构建请求头
    agent = 'Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0)
Gecko/20100101 Firefox/41.0'
    headers = {
        'User-Agent': agent,
        'Referer': forward_url
```

```
        }
        # 转发评论
        url = 'https://weibo.com/aj/v6/mblog/forward?ajwvr=
6&domain='+ domain +'&__rnd=' + (str(int(time.time() * 1000)))
        data = {
            'pic_src': '',
            'pic_id': '',
            'appkey': '',
            'mid': mid_list[0],
            'style_type': '1',
            'mark': '',
            'reason': reason,
            'location': location,
            'pdetail': page_id,
            'module': '',
            'page_module_id': '',
            'refer_sort': '',
            'is_comment_base': '1',
            'rank': '0',
            'rankid': '',
            '_t': '0'
        }
        r = session.post(url, data=data, headers=headers)
        # 根据返回内容判断是否成功
        if (r.json()['code']) == '100000':
            return ('转发成功')
        else:
            return ('转发失败')
```

转发评论功能定义函数 forward_weibo()：函数参数 session 是会话对象；forward_url 是被转发评论用户的首页链接；reason 是转发的评论内容。函数的功能逻辑与点赞功能大致相同，此处不做详细讲解。

将上述函数 like_weibo() 和 forward_weibo() 保存在文件 weibo_forward.py 中，代码与 16.6 节的运行方式一样，打开修改微博登录文件 weibo_login.py，代码如下：

```
    if __name__ == "__main__":
        # 构造请求头
```

```
        agent = 'Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0)
Gecko/20100101 Firefox/41.0'
        headers = {
            'User-Agent': agent
        }
        # 代理 IP
        proxies = {}
        # 新建会话
        session = requests.session()
        # 第三方平台账号、密码
        yundama_username = 'xxxx'
        yundama_password = 'xxxx'
        user_info = login('13435423143','xxxx')
        # 导入点赞和转发评论模块
        from weibo_forward import forward_weibo, like_weibo
        url = 'https://weibo.com/u17t'
        # 点赞
        result = like_weibo(session, url)
        print(result)
        # 转发评论
        result = forward_weibo(session, url, 'Python 网络爬虫 ')
        print(result)
```

16.8 本章小结

通过本章的学习，读者要着重掌握以下知识点：

1. 项目实现的功能

- weibo_login.py：微博用户登录，同时也是程序运行文件。
- weibo_verify_code.py：第三方平台 API，实现验证码识别。
- weibo_collect.py：根据关键字搜索并采集热门微博。
- weibo_send.py：发布微博。
- weibo_follow.py：关注用户。
- weibo_forward.py：微博点赞和转发评论。

- data.csv：存储采集数据。
- 文件夹 video 和 image：分别存储采集的视频和图片。

2. 微博登录实现难点

（1）账号密码的加密处理。加密方法一般在 JS 代码中能直接找到，开发人员需要对 JS 代码解读分析。

（2）带验证码登录和普通登录的区别，程序运行要根据当前的登录模式而做出响应的登录处理。

（3）用户登录成功后获取用户信息。

（4）第三方平台识别验证码。

3. 关键字搜索热门微博实现难点

（1）关键字 URL 编码处理，关键字进行了两次 URL 编码处理。

（2）响应内容乱码问题，需要对响应内容重新编码处理。

（3）文字过长的特殊处理。

（4）多线程下载图片和视频。

4. 发布微博实现难点

（1）图片上传分析以及功能实现。

（2）分析三种微博发布方式的异同。

5. 关注用户、点赞和转发评论实现难点

（1）分析请求参数含义以及来源。

（2）构建请求头。

第 17 章

Scrapy 爬虫框架

17.1 爬虫框架

爬虫框架是为解决爬虫问题而设计的具有一定约束性的支撑结构。在此结构上，可以根据具体问题扩展、安插更多的组成部分，从而更迅速和方便地构建完整的解决问题的方案。

Python 常见的爬虫框架如下。

- Scrapy 框架：Scrapy 框架是一套比较成熟的 Python 爬虫框架，是使用 Python 开发的快速、高层次的信息爬取框架，可以高效地爬取 Web 页面并提取出结构化数据。
- PySpider 框架：PySpider 是以 Python 脚本驱动的抓取环模型爬虫框架。

- Crawley 框架：Crawley 也是 Python 开发的爬虫框架，该框架致力于改变人们从互联网中提取数据的方式。
- Portia 框架：Portia 是一款允许没有任何编程基础的用户可视化地爬取网页的爬虫框架。
- Newspaper 框架：Newspaper 是一款用来提取新闻、文章以及内容分析的 Python 爬虫框架。

爬虫框架能为项目开发起到规范作用，也因为如此，使其失去一定的灵活性。很多人会将 Requests 和 Scrapy 两者进行对比，前者是第三方库，后者是爬虫开发框架，尽管两者不在同一层次，但还是有一定的对比性。

- 规范性：Scrapy 有自身的一套规则，自带功能模块能完成爬虫开发，各个功能代码划分明确。Requests 只规范数据爬取，不支持数据清洗和数据存储，需结合其他库一起使用才能完成爬虫开发。
- 灵活性：Scrapy 有较强的规范性，导致其灵活性比不上 Requests，对于一些设计不合理的网站或者较为特殊的网站，Requests 能针对其特殊性制定完善的解决方案。这方面对比 Scrapy 具有一定优势。
- 适用范围：Scrapy 适用于大型爬虫开发项目，主要归功于其具有明确的规范性，便于开发者对代码的维护和管理。Requests 对开发人员的编程习惯有较大影响，如果架构设计不合理或者替换开发人员，会使代码维护管理难以把控。

总地来说，无论是框架式开发还是非框架开发，都应针对项目的整体需求制定合理的开发设计方案。只要是合理的便是最好的，无论是框架与非框架，只是一个开发工具而已。

在 Python 中，开源爬虫框架很多，但并不需要掌握每一种爬虫框架，只需要深入掌握一种即可。大部分爬虫框架的实现方式都大同小异，基本上都是围绕爬虫开发流程（网页抓取、数据清洗、数据入库和异步并发处理等方面）设计而成的。

Scrapy 是一个为了爬取网站数据、提取结构性数据而编写的应用框架，主要应用在数据挖掘、信息处理或存储历史数据等一系列程序中。Scrapy 最初是为了页面抓取所设计的，也可以应用在获取 API 所返回的数据（例如 Amazon Associates Web Services）或者通用的网络爬虫中。

Scrapy 基于 Twisted 架构，使得可以级联多个操作，包括清理、组织、存储数据到数据库等。假设抓取一个网站，网站的每一页有上百数据，Scrapy 可以同时对这个网站发起 16 个或者更多请求，假如每个请求需要一秒钟来完成，相当于每秒钟爬取 16 个页面，每秒钟生成 1600 条数据，把这些数据同时存储入库，每条数据的存储需要 3 秒钟（假设时间），为了处理这 16 个请求，就需要运行 1600×3 = 4800 个并发的写入请求，对于一个传统的多线程程序来说，就需要转换成 4800 个线程，这会对系统造成极大的压力。对于 Scrapy 来说，只要硬件过关，4800 个并发请求是没有问题的。除此之外，还提供 selectors（在 lxml 的基础上提供了更高级的接口），可以高效地处理不完整的 HTML 代码。

17.2 Scrapy 的运行机制

Scrapy 使用 Twisted 异步网络库来处理网络通信，架构清晰，并且包含各种中间件接口，可以灵活地完成各种需求。Scrapy 的整体架构如图 17-1 所示。

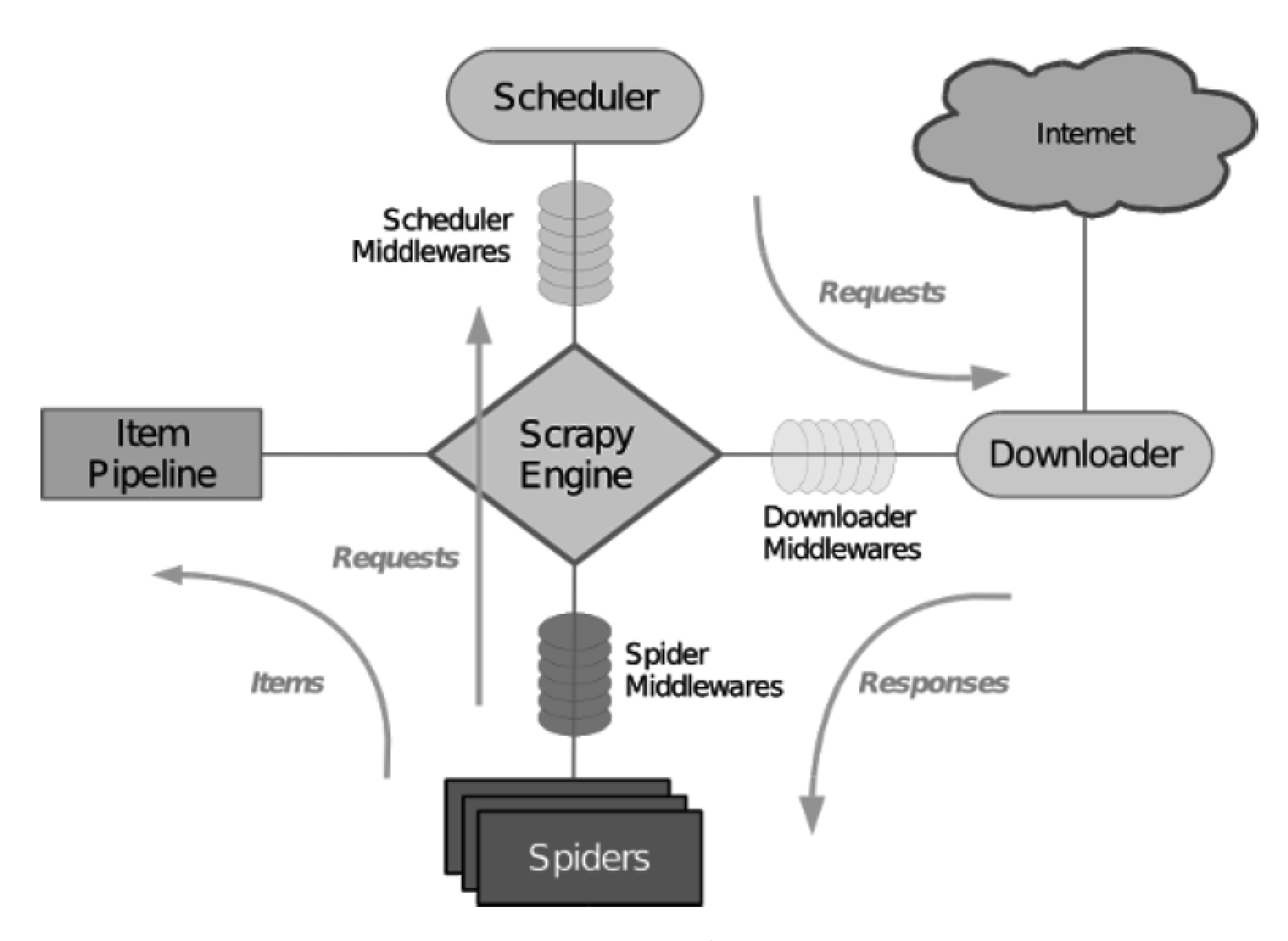


图 17-1　Scrapy 的运行机制

Scrapy 的运行机制大概如下：

（1）引擎从调度器中取出一个 URL（URL），用于接下来的抓取。

（2）引擎把 URL 封装成请求（Request）传给下载器，下载器把资源下载后封装成应答包（Response）。

（3）爬虫解析 Response。

（4）若解析出实体（Item），则交给实体管道进行进一步的处理。

（5）若解析出的是 URL，则把 URL 交给 Scheduler 等待抓取。

Scrapy 运行离不开各个组件相互合作和调度。结合图 17-1，各个组件的功能说明如下。

- 引擎（Scrapy）：处理整个系统的数据流，触发事务（框架核心）。
- 调度器（Scheduler）：接受引擎发过来的请求，压入队列中，并在引擎再次请求的时候返回。
- 下载器（Downloader）：用于下载网页内容，并将网页内容返回给蜘蛛（Scrapy 下载器的运行原理是基于 Twisted 框架实现的）。
- 爬虫（Spiders）：从特定的网页中提取自己需要的信息，即实体（Item）。也可以从中提取出 URL，让 Scrapy 继续抓取下一个页面。
- 项目管道（Item Pipeline）：负责处理爬虫从网页中抽取的实体，主要的功能是持久化实体、验证实体的有效性、清除不需要的信息。当页面被爬虫解析后，将被发送到项目管道，并经过几个特定的次序处理数据。
- 下载器中间件（Downloader Middlewares）：位于 Scrapy 引擎和下载器之间的框架，处理引擎与下载器之间的请求及响应。
- 爬虫中间件（Spider Middlewares）：介于 Scrapy 引擎和爬虫之间的框架，主要工作是处理蜘蛛的响应输入和请求输出。
- 调度中间件（Scheduler Middewares）：介于 Scrapy 引擎和调度之间的中间件，从 Scrapy 引擎发送到调度的请求和响应。

17.3 安装 Scrapy

在安装 Scrapy 之前，需要先安装 Twisted。Twisted 可以使用 pip 安装，但使

用 pip 安装很容易出现错误，建议下载 Twisted 的 whl 文件安装（www.lfd.uci.edu/~gohlke/pythonlibs/），如图 17-2 所示。

Twisted, an event-driven networking engine.
- Twisted-17.9.0-cp27-cp27m-win32.whl
- Twisted-17.9.0-cp27-cp27m-win_amd64.whl
- Twisted-17.9.0-cp34-cp34m-win32.whl
- Twisted-17.9.0-cp34-cp34m-win_amd64.whl
- Twisted-17.9.0-cp35-cp35m-win32.whl
- Twisted-17.9.0-cp35-cp35m-win_amd64.whl
- Twisted-17.9.0-cp36-cp36m-win32.whl
- Twisted-17.9.0-cp36-cp36m-win_amd64.whl

图 17-2　Twisted 版本信息

下载安装包时，应根据系统选择对应的版本信息，如 Twisted-17.9.0-cp35-cp35m-win_amd64.whl，cp35 是 Python 3.5 版本，amd64 代表 64 位系统。下载文件后保存在 E 盘，然后打开 CMD 窗口，将路径切换到 E 盘，输入安装指令：

```
E:\>pip install Twisted-17.9.0-cp35-cp35m-win_amd64.whl
```

完成 Twisted 安装后，可使用 pip 安装 Scrapy，安装指令如下：

```
pip install Scrapy
```

值得注意的是，最好先安装 Twisted，再安装 Scrapy。如果直接安装 Scrapy，在安装过程中就会出现报错信息：

```
building 'twisted.test.raiser' extension
error: Microsoft Visual C++ 14.0 is required. Get it with
"Microsoft Visual C++ Build Tools": http://landinghub.visualstudio.
com/visual-cpp-build-tools
```

如果出现上述报错信息，用户先安装 Twisted，再重新安装 Scrapy 即可解决。完成 Scrapy 的安装后，打开 CMD 窗口并进入 Python 交互式命令行，输入以下代码检测是否安装成功：

```
>>> import scrapy
>>> scrapy.__version__
'1.4.0'
```

17.4 爬虫开发快速入门

本节通过一个简单的项目讲解如何使用 Scrapy 实现爬虫开发，以百度知道的问题列表为例。在浏览器中打开网页（https://zhidao.baidu.com/list?cid=110）和开发者工具，查找并分析网页数据的生成方式。最终在 Doc 标签下找到数据所在位置，分析得知每条数据在标签 <a> 中，而标签 <a> 嵌套在标签 <div> 中，class 属性的值为 question-title，如图 17-3 所示。

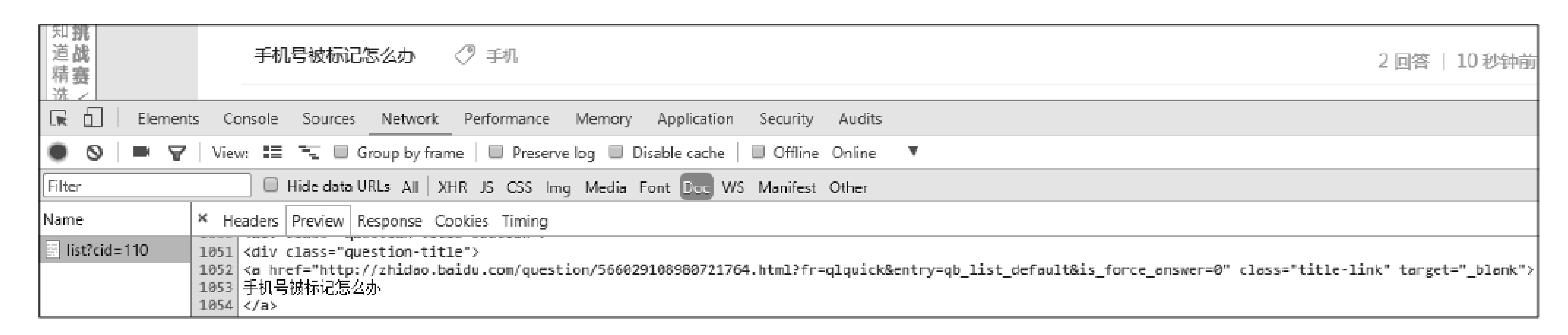


图 17-3 分析百度知道问题列表

根据简单分析，使用 Scrapy 完成上述开发需求。首先创建 Scrapy 项目，在 CDM（终端）下切换到 E 盘路径，本项目以“baidu”为项目名称，创建项目命令如下：

```
scrapy startproject baidu
```

创建项目后，可在 E 盘找到“baidu”文件夹，在 Pychram 下打开该文件夹，目录结构如图 17-4 所示。

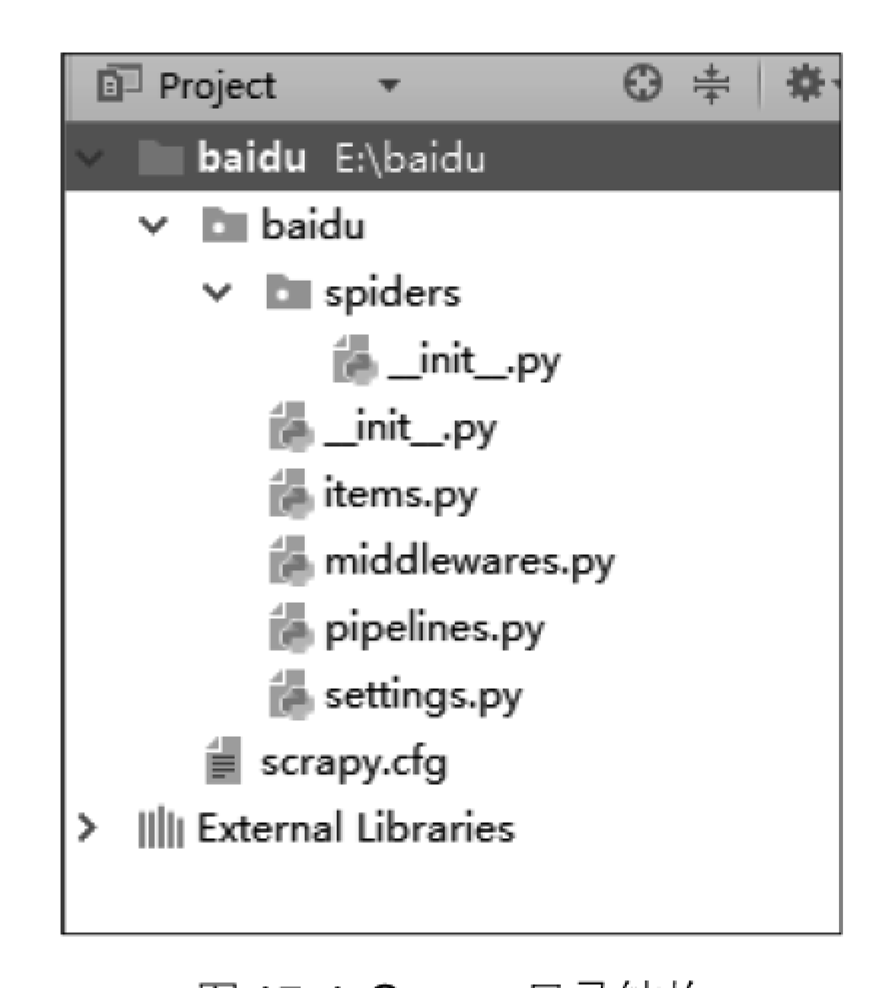


图 17-4 Scrapy 目录结构

项目文件说明如下。

- spiders（文件夹）：编写爬虫规则，实现数据爬取和数据清洗处理。
- items.py：数据定义和实例化，用于寄存清洗后的数据。
- middlewares.py：是介于 Scrapy 的 request/response 处理的钩子框架，用于全局修改 Scrapy request 和 response 的一个轻量、底层的系统。
- pipelines.py：执行保存数据的操作，数据对象来源于 items.py。
- setting.py：整个框架配置文件。
- scrapy.cfg：项目部署文件。

使用框架式开发一般都有功能实现次序，次序不是固定不变的，很大程度上根据开发人员的编程习惯所决定。Scrapy 常用功能实现次序如下。

- setting.py：主要配置爬虫信息，如请求头、中间件和延时设置等。
- items.py：定义存储数据对象，主要衔接 spiders（文件夹）和 pipelines.py。
- pipelines.py：数据存储，数据格式以字典形式表现，字典的键是 items.py 定义的变量。
- spiders（文件夹）：编写爬虫规则。

下面按照上述实现次序讲解功能代码的编写。

步骤 01 打开 setting.py，发现文件大部分内容已被注释，注释内容有配置代码、配置说明和相应的官方文档链接。本项目只需设置 Item Pipeline 和请求头即可，找到以下代码，将其注释去掉，其余代码不做任何操作：

```
# 指定数据入库的函数
ITEM_PIPELINES = {
   'baidu.pipelines.BaiduPipeline': 300,
}
# 设置请求头
DEFAULT_REQUEST_HEADERS = {
  'Accept': 'text/html,application/xhtml+xml,application/
xml;q=0.9,*/*;q=0.8',
  'Accept-Language': 'en',
}
```

配置信息说明如下：

- ITEM_PIPELINES 用于激活 pipelines.py 文件里的 BaiduPipeline 类，作用是告诉 Scrapy 在执行数据存储的时候使用哪个类对象实现存储。BaiduPipeline 是 Scrapy 项目自动生成的类，开发者也可根据实际需求添加或删除配置内容。
- DEFAULT_REQUEST_HEADERS 用于激活请求头，当 Scrapy 向网站发送请求的时候，如果该请求没有指明请求头内容，就默认使用该配置作为这个请求的请求头。

步骤 02 打开 items.py，Scrapy 已生成相关的代码及文档说明，开发者只需在此基础上定义类属性即可。本项目定义类属性 TitleName，代表问题列表中每条问题的内容。scrapy.Field() 是 Scrapy 的特有对象，其主要作用是处理并兼容不同的数据格式，开发者在定义类属性时无须考虑爬取数据的数据格式，Scrapy 会对数据格式做相应处理。实现代码如下：

```
import scrapy
class BaiduItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
    TitleName = scrapy.Field()
    pass
```

步骤 03 打开 pipelines.py，Scrapy 已生成类 BaiduPipeline 和相关说明，类 BaiduPipeline 就是 setting.py 配置 ITEM_PIPELINES 的内容。数据存储主要在类方法 process_item() 中执行，本项目以将数据存储在文本文档为例进行介绍，代码如下：

```
class BaiduPipeline(object):
    def process_item(self, item, spider):
        # 参数 item 是 items.py 的对象
        # 以下代码自行编写
        file = open('E:\\data.txt', 'a')
        for i in item['TitleName']:
            value = i.replace("\n", "")
            file.write(value + "\r\n")
        file.close()
        # 以上代码自行编写
```

```
        # return 主要输出 item 内容，若不需要，则可注释掉
        return item
```

步骤 04 spiders（文件夹）用于编写爬虫规则，可以在已有的__init__.py 文件中编写具体的爬虫规则，但实际开发可能有多个爬虫规则，所以建议一个爬虫规则用一个文件表示，这样便于维护和管理。回到项目中，我们创建文件 Spider_spiders.py，代码如下：

```
# 导入 items.py 的 BaiduItem，存放爬取数据
from baidu.items import BaiduItem
# Scrapy 自带数据清洗模块
from scrapy.selector import Selector
# Scrapy 搜索引擎
from scrapy.spider import Spider
# 爬虫规则，一个爬虫以类为实现对象
class Baispider(Spider):
    # 属性 name 必须设置，而且是唯一命名的，用于运行爬虫
    name = "Baidu_know"
    # 设置允许访问域名
    allowed_domains = ["baidu.com"]
    # 设置 URL
    start_urls = [
        "https://zhidao.baidu.com/list?cid=110",
        "https://zhidao.baidu.com/list?cid=110102"
    ]
  # 函数 parse 处理响应内容，函数名不能更改。
    def parse(self, response):
        # 将响应内容生成 Selector，用于数据清洗
        sel = Selector(response)
        items = []
        # 定义 BaiduItem 对象
        item = BaiduItem()
        title = sel.xpath('//div[@class="question-title"]/a/
text()').extract()
        for i in title:
            items.append(i)
```

```
        item['TitleName'] = items
        return item
```

上述代码说明如下：

（1）爬虫规则以类为实现单位，并继承父类 Spider，Spider 是 Scrapy 的爬虫引擎之一。

（2）属性 name 不能为空，其是程序运行入口，如果有多个爬虫规则，那么每个规则的属性 name 不能重复，否则 Scrapy 无法识别执行哪一个爬虫规则。

（3）allowed_domains 是设置允许访问的域名，如果为空，就说明对域名不做访问限制。

（4）start_urls 用于设置爬取对象的 URL，程序运行时会对 start_urls 遍历处理。

（5）类方法 parse() 用于处理网站的响应内容，如果爬虫引擎是 Spider，方法名就不能更改。

完成上述功能开发后，使用 CMD（终端）启动程序，将路径切换到 E:\baidu，运行命令如下：

```
E:\baidu>scrapy crawl Baidu_know
```

scrapy crawl 是启动 Scrapy 的命令符，Baidu_know 是 Spider_spiders.py 文件的 Baispider 类属性 name。程序运行结果如图 17-5 所示。

从运行结果看出，程序对 start_urls 的 URL 遍历访问，并返回 None 对象。因为对 pipelines.py 的 return item 做了注释处理，如果去掉注释，就会返回爬取的数据内容。程序运行结束后，Scrapy 会将运行信息返回，开发人员可根据信息调整 setting.py 的并发数和延时配置。

Baispider 类继承自父类 Spider，在 Scrapy 中，大多数爬虫使用 Spider 类足以胜任，如果要爬取全站数据而且具有一定规则的网站，Spider 虽然可以实现，但实现过程相当复杂，这时我们需要更强大的武器 CrawlSpider。

```
n 127.0.0.1:6023
2017-12-11 17:48:17 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://zhida
o.baidu.com/robots.txt> (referer: None)
2017-12-11 17:48:17 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://zhida
o.baidu.com/list?cid=110> (referer: None)
2017-12-11 17:48:17 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://zhida
o.baidu.com/list?cid=110102> (referer: None)
2017-12-11 17:48:17 [scrapy.core.scraper] DEBUG: Scraped from <200 https://zhida
o.baidu.com/list?cid=110>
None
2017-12-11 17:48:17 [scrapy.core.scraper] DEBUG: Scraped from <200 https://zhida
o.baidu.com/list?cid=110102>
None
2017-12-11 17:48:17 [scrapy.core.engine] INFO: Closing spider (finished)
2017-12-11 17:48:17 [scrapy.statscollectors] INFO: Dumping Scrapy stats:
{'downloader/request_bytes': 789,
 'downloader/request_count': 3,
 'downloader/request_method_count/GET': 3,
 'downloader/response_bytes': 47129,
 'downloader/response_count': 3,
 'downloader/response_status_count/200': 3,
 'finish_reason': 'finished',
 'finish_time': datetime.datetime(2017, 12, 11, 9, 48, 17, 782106),
 'item_scraped_count': 2,
 'log_count/DEBUG': 6,
 'log_count/INFO': 7,
 'response_received_count': 3,
 'scheduler/dequeued': 2,
 'scheduler/dequeued/memory': 2,
 'scheduler/enqueued': 2,
 'scheduler/enqueued/memory': 2,
 'start_time': datetime.datetime(2017, 12, 11, 9, 48, 16, 876446)}
2017-12-11 17:48:17 [scrapy.core.engine] INFO: Spider closed (finished)
```

图 17-5 Scrapy 运行结果

CrawlSpider 也继承自父类 Spider，拥有父类 Spider 的全部属性，并有自身的独特属性。

（1）rules 是 Rule 对象的集合，用于匹配目标网站并排除干扰。

（2）parse_start_url 用于爬取起始响应，必须要返回 Item，Request 是其中之一。

以上述项目为例，使用 CrawlSpider 实现上述爬虫规则：首先在 spiders（文件夹）下创建文件 CrawlSpider_spiders.py，代码如下：

```
# 导入 items.py 的 BaiduItem，存放爬取数据
from baidu.items import BaiduItem
# Scrapy 自带数据清洗模块
from scrapy.selector import Selector
# 导入 CrawlSpider
from scrapy.contrib.spiders import CrawlSpider, Rule
from scrapy.contrib.linkextractors import LinkExtractor
# 爬虫规则，一个爬虫以类为实现对象
class Baispider(CrawlSpider):
```

```
    # 属性 name 必须设置，而且是唯一命名的，用于运行爬虫
        name = "Baidu"
    # 设置允许访问域名
        allowed_domains = ["baidu.com"]
    # 设置 URL
        start_urls = [
            "https://zhidao.baidu.com/list?cid=110"
            ]
    # 编写爬取规则
        rules = (
            Rule(LinkExtractor(allow=('zhidao.baidu.com/question/', ),
deny=(),), callback='parse_item'),
           )
    # 编写处理函数
        def parse_item(self, response):
            sel = Selector(response)
            items = []
            item = BaiduItem()
            title = sel.xpath('//span[@class="ask-title "]/text()').
extract()
            for i in title:
                items.append(i)
            item['TitleName'] = items
            return item
```

上述代码与 Spider_spiders.py 的实现功能是一致的，但在逻辑处理上完全不同：

（1）Spider_spiders.py 继承自 Spider，运行方式是遍历 start_urls 的 URL，从每个 URL 获取数据，数据主要来源于 start_urls 的 URL。

（2）CrawlSpider_spiders.py 继承自 CrawlSpider，start_urls 的 URL 被访问后，获取其响应内容里的 URL 列表，再根据 rules 规则对得到的 URL 列表进行筛选，选出所有符合规则的 URL，并对符合规则的 URL 调用 callback 所指定的函数进行访问和处理。

CrawlSpider 类的 Rule 参数说明如下。

- allow：满足括号中的值会被提取，如果为空，就全部匹配，支持正则表达式实现模糊匹配。

- deny：与匹配值不匹配的 URL 不提取。
- allowed_domains：会被提取的 URL 的 domains。
- deny_domains：一定不会被提取 URL 的 domains。
- callback：指定回调函数处理符合筛选规则的 URL 的响应内容。

从运行逻辑分析，CrawlSpider 爬虫更适合全站数据爬取和通用爬虫开发。因为 rules 是 Rule 对象的集合，如果需要编写多个规则，就可以设置多个 Rule 对象，callback 所指定的函数也可以自行命名。相对 Spider 来说，CrawlSpider 在使用上较为灵活一些。

17.5 Spiders 介绍

Spider 是定义如何抓取某个网站（或一组网站）的类，包括如何执行抓取（访问 URL）以及如何从页面中提取结构化数据（抓取数据）。换句话说，Spider 是开发者自定义的类，用于为特定网站（在某些情况下是一组网站）抓取和解析页面。

Spider 的执行周期如下：

（1）抓取第一个 URL 的初始请求，然后指定一个回调函数，从请求的响应来调用回调函数，请求链接通过调用 start_requests() 方法（该方法在默认情况下是 GET 方式），parse 方法作为回调函数处理请求链接返回的请求结果。

（2）在回调函数中，主要是解析响应（网页）内容，并将解析后的数据存储在 Item 对象中。如果解析的内容中需要产生多次请求，就可将 URL 传递给 Request 对象并指定某个回调函数，然后由 Scrapy 访问下载，通过指定的回调处理它们的响应。

（3）在回调函数中，通常使用选择器（也可以使用 BeautifulSoup、lxml 等第三方库）解析页面内容，并将解析的数据存储在 Item 对象中。

最后，从 Spider 返回的 Item 对象在 item pipeline 对象中进行数据存储。

Spider 的种类如下。

- scrapy.spiders.Spider：最简单的 Spider 类，其他的 Spider 也继承自该类（包括 Scrapy 其他定义的 Spider 以及开发者自定义的 Spider）。它不提供任何特

殊的功能，只提供一个默认的 start_requests() 方法，请求从 start_urls 开始，Spider 发送请求，并使用函数 parse 处理每个响应内容。

- scrapy.spiders.CrawlSpider：这是抓取常规网站最常用的 Spider，因其提供了一个方便的机制，可通过定义一组规则来跟踪 URL，适合全站数据爬取和通用爬虫开发。除了拥有 scrapy.spiders.Spider 全部属性之外，还有特定属性 rules 和 parse_start_url 方法。
- scrapy.spiders.XMLFeedSpider：用来爬取 XML 形式的网页内容，通过某个指定的节点来遍历。可使用 iternodes、xml 和 html 三种形式的迭代器，不过当内容比较多的时候，推荐使用 iternodes，可以节省内存、提升性能，不需要将整个 DOM 加载到内存中再解析，而使用 html 可以处理 XML 有格式错误的内容。
- scrapy.spiders.CSVFeedSpider：与 XMLFeedSpider 非常相似，其遍历 CSV 行数，在每个迭代中被调用的方法是 parse_row()。
- scrapy.spiders.SitemapSpider：SitemapSpider 通过使用 Sitemaps 发现网址并抓取网站，支持嵌套 Sitemap 和从 robots.txt 中发现 Sitemap 网址。这类爬虫主要用于搜索引擎开发，主要用于爬取整个网站的地图和全部 URL。

一般来说，目前大多数网站主要以 HTML 为主，Spiders 开发是以 Spider 和 CrawlSpider 为主，本书主要以这两者为讲述对象。

17.6 Spider 的编写

从 17.4 节的内容得知，Spider_spiders.py 爬虫规则的请求方式都是 GET 请求，但在实际开发中，我们还需要使用 POST 请求，那么如何在 Spider 中实现 POST 请求呢？

首先创建一个新的项目，命名为 mySpider：

```
scrapy startproject mySpider
```

在项目里的 spiders（文件夹）中创建文件 post_spiders.py，代码如下：

```
from scrapy.selector import Selector
from scrapy.spider import Spider
import scrapy
class Baispider(Spider):
```

```
    name = "Post_spider"
    allowed_domains = []
    start_urls = [
    "http://127.0.0.1:5000/",
    ]
    # 爬虫入口——重写 start_requests 方法
    # scrapy.FormRequest 是 POST 方式，formdata 是 POST 参数，callback 回调
函数
    def start_requests(self):
    return [scrapy.FormRequest(
    self.start_urls[0],
    formdata={"Python": " 爬虫开发 "},
    callback=self.mypsot)]
    # 回调函数
    def mypsot(self, response):
    data = Selector(response).xpath('//p/text()').extract()[0]
    print(data)
```

整个项目 mySpider 只添加上述代码和文件，其余文件不做修改和添加。为了方便测试代码，我们在本地使用 Flask 搭建一个测试系统（Flask 安装：pip install flask），系统代码如下：

```
from flask import Flask, request
app = Flask(__name__)
# app.route 设置 URL 路径，methods 是请求方式
# hello_world 视图函数
@app.route('/', methods=['POST','GET'])
def hello_world():
    # 判断请求方式，返回不同结果
    # POST 请求
    if request.method == 'POST':
        return "This is Post,your post data is " + request.
form['Python']
    # GET 请求
    else:
        return 'Hello World!'
# 系统启动运行
```

```
if __name__ == '__main__':
    app.run()
```

将系统代码保存在 system.py 文件中，然后运行文件即可启动系统。系统启动后，运行 mySpider 项目，运行结果如图 17-6 所示。

```
n 127.0.0.1:6023
2017-12-12 18:18:58 [scrapy.core.engine] DEBUG: Crawled (404) <GET http://127.0.
0.1:5000/robots.txt> (referer: None)
2017-12-12 18:18:58 [scrapy.downloadermiddlewares.retry] DEBUG: Retrying <POST h
ttp://127.0.0.1:5000/> (failed 1 times): [<twisted.python.failure.Failure twiste
d.internet.error.ConnectionDone: Connection was closed cleanly.>]
2017-12-12 18:18:58 [scrapy.core.engine] DEBUG: Crawled (200) <POST http://127.0
.0.1:5000/> (referer: None)
This is Post,your post data is 爬虫开发
2017-12-12 18:18:58 [scrapy.core.engine] INFO: Closing spider (finished)
2017-12-12 18:18:58 [scrapy.statscollectors] INFO: Dumping Scrapy stats:
{'downloader/exception_count': 1,
 'downloader/exception_type_count/twisted.web._newclient.ResponseNeverReceived':
 1,
 'downloader/request_bytes': 817,
 'downloader/request_count': 3,
 'downloader/request_method_count/GET': 1,
 'downloader/request_method_count/POST': 2,
 'downloader/response_bytes': 532,
 'downloader/response_count': 2,
 'downloader/response_status_count/200': 1,
 'downloader/response_status_count/404': 1,
 'finish_reason': 'finished',
 'finish_time': datetime.datetime(2017, 12, 12, 10, 18, 58, 125681),
 'log_count/DEBUG': 4,
```

图 17-6 post_spiders 运行结果

可以看到运行结果输出“This is Post,your post data is 爬虫开发”，返回结果和 Flask 系统代码是一致的，说明在 Scrapy 中可重写 start_requests 来改写初始请求方式。

一个完整的爬虫会将 POST 和 GET 请求相互交错使用，而且每个请求都可能需要特定的请求头和 Cookies。以 mySpider 项目为例实现上述功能需求：在 mySpider 项目的 spiders（文件夹）下新建文件 get_post_spiders.py，代码如下：

```
from scrapy.selector import Selector
from scrapy.spider import Spider
import scrapy

class Baispider(Spider):
    name = "Get_Post_spider"
    allowed_domains = []
    start_urls = [
        "http://127.0.0.1:5000/",
    ]
```

```
        # 定义请求头和 Cookies，两者皆以字典形式表示
        headers = {'User-Agent':
                       'Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0)
Gecko/20100101 Firefox/41.0',
                   }
        cookies = {}

        # 处理第一次 GET 请求的响应内容，return 用于发送第二次 POST 请求
        def parse(self, response):
            data = Selector(response).xpath('//p/text()').extract()[0]
            print(data)
            return [scrapy.FormRequest(
                self.start_urls[0],
                cookies=self.cookies,
                headers=self.headers,
                formdata={"Python": " 爬虫开发 "},
                callback=self.mypsot)]

        # 处理第二次 POST 请求的响应内容，return 用于发送第三次 GET 请求
        def mypsot(self, response):
            data = Selector(response).xpath('//p/text()').extract()
[0]
            print(data)
            return scrapy.Request(self.start_urls[0], cookies=self.
cookies,
                                  headers=self.headers,
callback=self.myget)

        # 处理第三次 GET 请求的响应内容
        def myget(self, response):
            data = Selector(response).xpath('//p/text()').extract()[0]
            print(data)
```

从上述代码分析三次请求：

第一次 GET 请求是 Scrapy 默认 start_requests 实现的，回调函数是 parse。

第二次 POST 请求是在函数 parse 处理完第一次请求的响应内容后，通过 return 发送第二次请求，并设置请求头和 Cookies，回调函数是 mypsot。

第三次 GET 请求是在函数 mypsot 处理完第二次请求的响应内容后，通过 return 发送第三次请求，并设置请求头和 Cookies，回调函数是 myget。

打开 CMD 窗口，运行 get_post_spiders.py 的爬虫规则，运行结果如图 17-7 所示。

```
2017-12-13 09:47:06 [scrapy.core.engine] INFO: Spider opened
2017-12-13 09:47:06 [scrapy.extensions.logstats] INFO: Crawled 0 pages (at 0 pag
es/min), scraped 0 items (at 0 items/min)
2017-12-13 09:47:06 [scrapy.extensions.telnet] DEBUG: Telnet console listening o
n 127.0.0.1:6023
2017-12-13 09:47:06 [scrapy.core.engine] DEBUG: Crawled (404) <GET http://127.0.
0.1:5000/robots.txt> (referer: None)
2017-12-13 09:47:06 [scrapy.core.engine] DEBUG: Crawled (200) <GET http://127.0.
0.1:5000/> (referer: None)
Hello World!
2017-12-13 09:47:06 [scrapy.core.engine] DEBUG: Crawled (200) <POST http://127.0
.0.1:5000/> (referer: http://127.0.0.1:5000/)
This is Post,your post data is 爬虫开发
2017-12-13 09:47:07 [scrapy.core.engine] DEBUG: Crawled (200) <GET http://127.0.
0.1:5000/> (referer: http://127.0.0.1:5000/)
Hello World!
2017-12-13 09:47:07 [scrapy.core.engine] INFO: Closing spider (finished)
2017-12-13 09:47:07 [scrapy.statscollectors] INFO: Dumping Scrapy stats:
```

图 17-7 get_post_spiders 运行结果

此外，Spiders 中的 CrawlSpider、XMLFeedSpider、CSVFeedSpider 和 SitemapSpider 都继承于父类 Spider，因此前面实现的功能都适用于 Spiders 所有类。

17.7 Items 的编写

数据抓取的主要目标是从非结构化来源（通常是网页）中提取结构化数据。Scrapy 可以将提取的数据作为 Python 字典返回，但 Python 字典缺乏结构，字典的键会在输入时出现拼写错误或者返回数据不一致，因此，Scrapy 提供了 Items 对象，用于管理和规范爬取数据，使其结构规范化。

Items 主要存放在项目文件 items.py 中，每个 Items 对象以类的形式声明和命名，类属性为 Items 的字段，也就是需要存储数据的元数据键（metadata key）。Items 可脱离 Scrapy 项目单独使用，为了更好演练，在 E 盘下创建文件 items.py，代码如下：

```
import scrapy
# Product 类继承自 Item 类
class Product(scrapy.Item):
```

```
        name = scrapy.Field()
        price = scrapy.Field()
        stock = scrapy.Field()
        # last_updated 指明了该字段的序列化函数
        last_updated = scrapy.Field(serializer=str)

if __name__=="__main__":
product = Product(name='Desktop PC', price=1000)
print (product)
```

使用代码定义 Product 类，类属性 name、price、stock 和 last_updated 分别代表产品名称、价格、库存数和更新时间。Scrapy 声明字段无须考虑其数据类型，统一以 scrapy.Field() 命名即可。运行 items.py 文件，输出结果如下：

```
{'name': 'Desktop PC', 'price': 1000}
```

除此之外，还可以对其进行读取和判断等操作。代码如下：

```
# 数据存储一
product = Product(name='Desktop PC',price=1000)
print (product)
# 数据存储二
item = Product()
item['name'] = 'Mac'
item['price'] = 2000
print(item)
# 读取数据内容一，若不存在，则会输出 None
print(item.get('name', 'None'))
print(item.get('stock', 'None'))
# 读取数据内容二，使用该方法读取，若不存在，则会提示 keyerror
print(item['name'])
# print(item['stock'])
# 判断是否存在字段，输出 True 或 False
print('name' in item)
print('stock' in item)
# 获取键值对
print(item.keys())
print(item.items())
```

17.8 Item Pipeline 的编写

当 Spiders 爬取的数据存放到 Items 之后，回调函数的 return（yield）返回 Items 对象，这时会触发 Item Pipeline 对 Items 对象的操作。Item Pipeline 主要存放在项目文件 pipelines.py 中，用于实现数据存储。以 17.4 节的项目为例，我们将数据存储介质由文本文档改为 MongoDB，MongoDB 的数据库结构信息如图 17-8 所示。

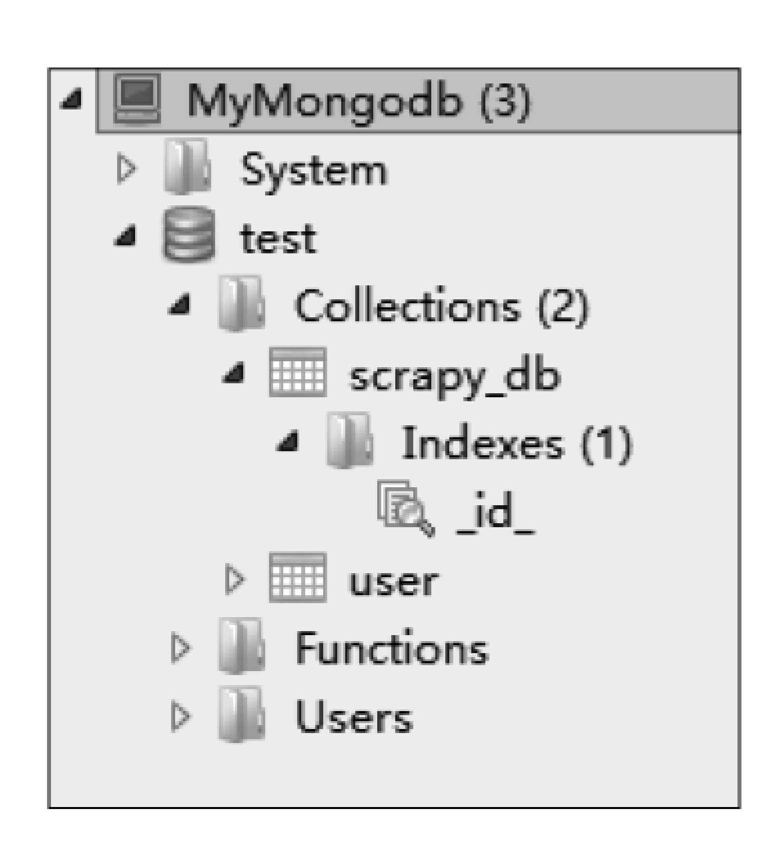


图 17-8 MongoDB 数据库

数据库信息如下：

（1）数据库所在服务器的 IP 地址：localhost。

（2）数据库端口：27017。

（3）数据库名：testCollection 名称：scrapy_db。

将数据库信息写入配置文件 setting.py，在 setting.py 中添加以下代码：

```
ITEM_PIPELINES = {
   'baidu.pipelines.BaiduPipeline': 300,
}
# 数据库 IP
MONGODB_SERVER = "localhost"
# 端口
MONGODB_PORT = 27017
# Database 名称
MONGODB_DB = "test"
# Collection 名称
MONGODB_COLLECTION = "scrapy_db"
```

完成数据库信息配置，接着编写 Item Pipeline 功能代码，pipelines.py 的代码如下：

```
# 导入 pymongo
from pymongo import MongoClient
# 导入 setting 配置信息
from scrapy.conf import settings
```

```
class BaiduPipeline(object):
    def __init__(self):
  # 连接数据库
        connection = MongoClient(
            settings['MONGODB_SERVER'],
            settings['MONGODB_PORT']
        )
        db = connection[settings['MONGODB_DB']]
        self.collection = db[settings['MONGODB_COLLECTION']]

    def process_item(self, item, spider):
  # 入库处理
        self.collection.insert(dict(item))
        return item
```

以 17.4 节的 Spider_spiders.py 爬虫规则运行程序，并查看数据库的数据信息，如图 17-9 所示。

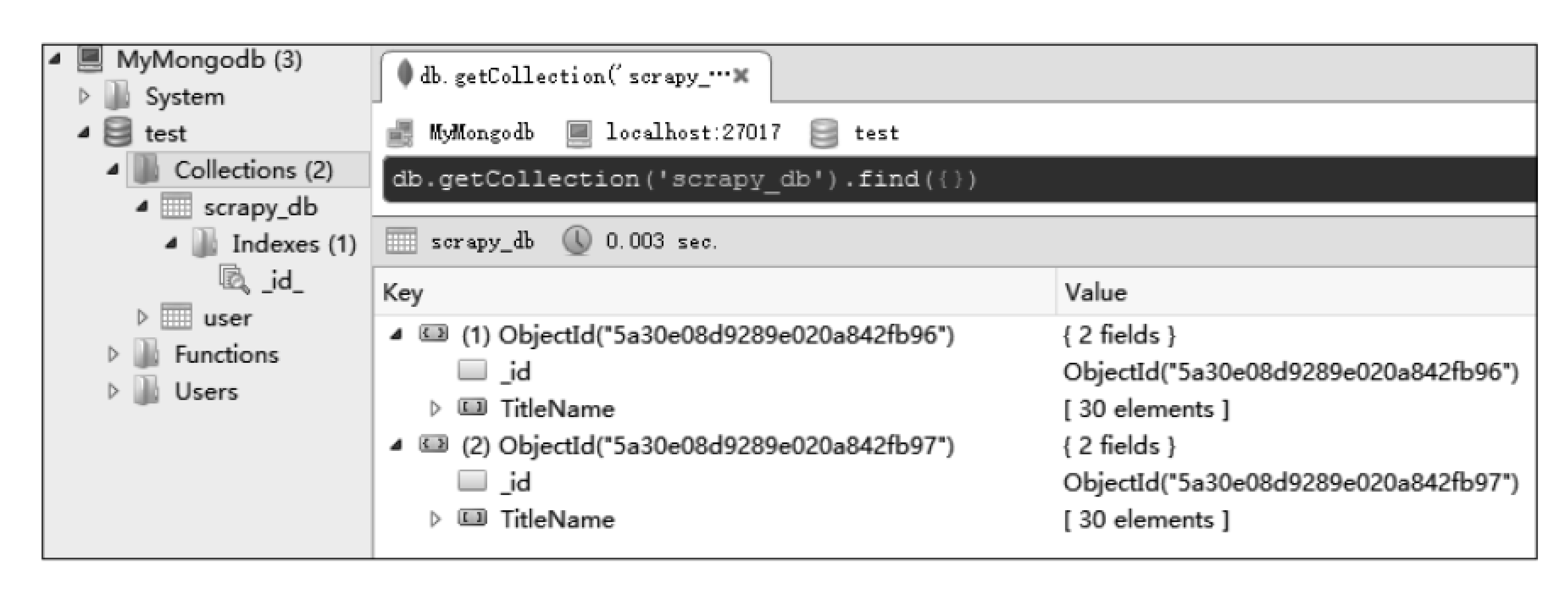


图 17-9 MongoDB 入库结果

从入库结果看到，生成了两条文档，每条文档对应 Spider_spiders.py 中 start_urls 的数量；每条 URL 有 30 条数据，也符合字段 TitleName 的数据量。从代码分析，在类 BaiduPipeline 的初始（__init__）函数中实现数据库连接功能，在函数 process_item 中实现数据入库。

上述功能实现 MongoDB 入库，若要使用 SQLAlchemy 实现数据入库，则实现方式与上述大致相同。同样以 17.4 节的项目为例，以 MySQL 数据库为存储对象，MySQL 数据库信息如下：

（1）数据库所在服务器的 IP 地址：localhost。

（2）数据库用户：root。

（3）数据库密码：1234。

（4）数据库名：test。

将数据库信息写入配置文件 setting.py，在 setting.py 中添加代码如下：

```
# SQLAlchemy 连接数据库
MYSQL_CONNECTION = 'mysql+pymysql://root:1234@localhost/
test?charset=utf8'
```

编写 Item Pipeline 功能代码，pipelines.py 的代码如下：

```
# 导入 SQLAlchemy
from sqlalchemy import *
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base
# 导入 setting 配置信息
from scrapy.conf import settings

# 定义映射类
Base = declarative_base()
class scrapy_db(Base):
    __tablename__ = 'scrapy_db'
    id = Column(Integer(), primary_key=True)
    TitleName = Column(String(200))

class BaiduPipeline(object):
    def __init__(self):
        # 初始化，连接数据库
        conntion = settings['MYSQL_CONNECTION']
        engine = create_engine(conntion, echo=False,
                                         pool_size=2000)
        DBSession = sessionmaker(bind=engine)
        self.SQLsession = DBSession()
```

```
        # 创建数据表
        Base.metadata.create_all(engine)

    def process_item(self, item, spider):
        # 入库处理
        self.SQLsession.execute(scrapy_db.__table__.insert(),
                    [{'TitleName': i} for i in item['TitleName']])
        self.SQLsession.commit()
        return item
```

以 17.4 节的 Spider_spiders.py 爬虫规则运行程序，并查看数据库的数据信息，如图 17-10 所示。



图 17-10 SQLAlchemy 入库结果

从图 17-10 得知，入库数据量和使用 MongoDB 入库的数据量是一致的。无论使用关系式数据库还是非关系式数据库，数据入库逻辑都是相同的，根据数据入库逻辑总结 Item Pipeline 编写规则如下：

（1）使用 setting.py 配置数据库信息。数据库信息最好在 setting.py 中配置，这符合统一规范化开发要求。

（2）对 pipelines.py 的类初始化（__init__）函数实现数据库连接。如果使用 SQLAlchemy 入库，那么还需创建映射类映射数据表。

（3）由函数 process_item 实现数据入库。

17.9 Selectors 的编写

当抓取网页时，最常见的任务是从 HTML 源码中提取数据。Scrapy 提取数据有一套机制，被称作选择器（Seletors），通过特定的 XPath 或者 CSS 表达式来选择 HTML 中的某部分数据。当然，lxml 和 BeautifulSoup 也可以在 Scrapy 中担任数据清洗角色。

Scrapy 选择器主要用于爬虫规则的编写。以 17.4 节的 Spider_spiders.py 爬虫规则为例进行介绍：

```
# 导入 items.py 的 BaiduItem，存放爬取数据
from baidu.items import BaiduItem
# Scrapy 自带数据清洗模块
from scrapy.selector import Selector
# Scrapy 搜索引擎
from scrapy.spider import Spider
# 爬虫规则，一个爬虫以类为单位

class Baispider(Spider):
    # 属性 name 必须设置，而且是唯一命名的，用于运行爬虫
    name = "Baidu_know"
    # 设置允许访问域名
    allowed_domains = ["baidu.com"]
    # 设置 URL
    start_urls = [
        "https://zhidao.baidu.com/list?cid=110",
        "https://zhidao.baidu.com/list?cid=110102"
    ]
    # 函数 parse 处理响应内容，函数名不能更改
    def parse(self, response):
        # 将响应内容生成 Selector，用于数据清洗
        sel = Selector(response)
        items = []
        # 定义 BaiduItem 对象
        item = BaiduItem()
        title = sel.xpath('//div[@class="question-title"]/a/
text()').extract()
```

```
        for i in title:
            items.append(i)
        item['TitleName'] = items
        return item
```

从上述代码可知，选择器的使用步骤如下：

（1）from scrapy.selector import Selector：导入 Selector 对象。

（2）sel = Selector(response)：声明 Selector 对象，并将响应内容加载该对象中。

（3）sel.xpath(XPath 语法).extract()：使用 XPath 对数据进行清洗，方法 extract() 将数据以列表形式返回。

XPath 是一门用来在 XML 文件中选择节点的语言，也可以用在 HTML 中。CSS 是一门将 HTML 文档样式化的语言，选择器由它定义，并与特定 HTML 元素的样式相关。在两者的使用上，大部分开发人员偏向于 XPath。选择器主要掌握 XPath 或者 CSS 语法编写规则，本书以 XPath 语法为讲述重点。

XPath 使用路径表达式来选取 XML 文档中的节点或节点集，节点是通过沿着路径（path）或者步（steps）来选取的，使用 XPath 获取数据主要是找到数据所在的路径。例子如下：

```
<html>
 <head>
  <base href='http://example.com/' />
  <title>Example website</title>
 </head>
 <body>
  <div id='images'>
   <a href='image1.html'>Name: My image 1 <br /><img src='image1_
thumb.jpg' /></a>
   <a href='image2.html'>Name: My image 2 <br /><img src='image2_
thumb.jpg' /></a>
   <a href='image3.html'>Name: My image 3 <br /><img src='image3_
thumb.jpg' /></a>
   <a href='image4.html'>Name: My image 4 <br /><img src='image4_
thumb.jpg' /></a>
```

```
        <a href='image5.html'>Name: My image 5 <br /><img src='image5_
thumb.jpg' /></a>
      </div>
    </body>
  </html>
```

根据上述例子获取标签 <a>，href 属性为 image1.html 的内容，Xpath 语法如下：

```
    xpath('//div[@id="images"]/a[@href="image1.html]"/text()').
extract()
```

或者：

```
    xpath('//a[@href="image1.html]"/text()').extract()
```

对于上述两种不同的定位方法，说明如下：

（1）第一种方法是因为标签 <a> 嵌套在 <div> 中，所以先通过 //div[@id="images"] 对 <div> 定位，在已定位 <div> 的基础上添加 /a[@href="image1.html]，说明先查找 <div>，再查找 <div> 里面的 <a>。

（2）第二种方法是因为标签 <a> 的 href 属性为 image1.html，在整段 HTML 中具有唯一性，所以直接对标签 <a> 定位即可。

上述例子使用"//""/"和"@"这类特殊符号，这是 XPath 的路径表达式，常用的 XPath 路径表达式如表 17-1 所示。

表17-1 XPath路径表达式

表达式	描 述
nodename	选取此节点的所有子节点
/	从根节点选取
//	从匹配选择的当前节点选择文档中的节点，而不考虑它们的位置
.	选取当前节点
..	选取当前节点的父节点
@	选取属性

除了路径表达式外，XPath 的方括号（[]）可嵌套谓语（用来查找某个特定的节点或者包含某个指定值的节点）。简单地说，方括号中可编写标签的特性，从而精确

地找出所需的数据，如表 17-2 所示。

表17-2 路径表达式及结果

路径表达式	结果
/div/a[1]	选取属于div的第一个a标签
/div/a[last()]	选取div里最后的a标签
/div/a[last()-1]	选取div里倒数第二个a标签
/div/a[position()<3]	选取div前两个a标签
/div/a[@id!="image1.html"]	选取div里属性id不为image1.html的a标签

XPath 的定位与 Windows 系统的路径相同，但计算机上同一目录不允许存在同名的文件或文件夹，而 HTML 可存在这种情况，为了解决这个问题，XPath 可对标签的属性进行筛选，若有多个同时符合的条件，则会获取全部符合条件的数据。

17.10 文件下载

爬虫除了爬取数据之外，还常常需要爬取文件，如图片、文本文件和音视频等。Scrapy 在下载图片（文件）时提供了一个可重用的 Item Pipelines，称为 Media Pipeline。Media Pipeline 分为 Files Pipeline 和 Images Pipeline。

Files Pipeline 和 Images Pipeline 实现的功能如下：

（1）能避免重新下载已下载过的数据。

（2）可以指定下载后保存的路径。

Images Pipeline 为处理图片提供了额外的功能：

（1）将所有下载的图片格式转换成普通的 JPEG 并使用 RGB 颜色模式。

（2）生成缩略图。

（3）检查图片的宽度和高度，确保它们满足最小的尺寸限制。

Pipeline 同时会在内部保存一个被调度下载的 URL 列表，然后将包含相同媒体的链接关联到这个队列上来，从而防止重复下载。

使用 Files Pipeline 实现下载的步骤如下：

步骤01 在 Spider 中爬取一个 Item 后，将相应的文件 URL 放入 file_urls 字段中。

步骤02 Item 被返回之后就会转交给 Item Pipeline。

步骤03 当这个 Item 到达 FilesPipeline 时，在 file_urls 字段中的 URL 列表会通过标准的 Scrapy 调度器和下载器来调度下载，并且优先级很高，在抓取其他页面前就被处理。而这个 Item 会一直在这个 Pipeline 中被锁定，直到所有的文件下载完成。

步骤04 当文件被下载完之后，结果会被赋值给另一个 files 字段。这个字段包含一个关于下载文件的新字典列表，比如下载路径、源地址、文件校验码。files 里面的顺序和 file_url 的顺序是一致的。若下载出错，则不会出现在这个 files 中。

ImagesPipeline 的使用跟 FilesPipeline 差不多，不过使用的字段名不一样，image_urls 用于保存图片 URL 地址，使用 ImagesPipeline 的好处是可以通过配置来提供额外的功能，比如生成文件缩略图、通过图片大小过滤需要下载的图片等。ImagesPipeline 使用 Pillow 来生成缩略图以及转换成标准的 JPEG/RGB 格式。

为了进一步掌握 Scrapy 下载功能，分别找出三个文件下载链接：

```
# 下载 zip 压缩包
'http://d.1.didiwl.com/PYTHON_zryycl.zip',
# 下载图片
'http://i0.hdslb.com/bfs/archive/9a8f816fdadd1b814c5ce51e7ead2531
9166eb92.jpg',
# 下载歌曲文件
'http://ws.stream.qqmusic.qq.com/C100001IqoFr2rNsGH.
m4a?fromtag=38'
```

创建新的 Scrapy 项目，名为 scrapy_download，创建命令如下：

```
scrapy startproject scrapy_download
```

创建项目后，打开 setting.py 文件，添加以下代码：

```
ITEM_PIPELINES = {
   'scrapy_download.pipelines.ScrapyDownloadPipeline': 300,
   'scrapy_download.pipelines.DownloadFlie': 1,
```

```
    }
    # 设置保存路径
    FILES_STORE = 'E:\\full\\'
    # 设置请求头
    DEFAULT_REQUEST_HEADERS = {
      'Accept': 'text/html,application/xhtml+xml,application/
xml;q=0.9,*/*;q=0.8',
      'User-Agent': 'Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0)
Gecko/20100101 Firefox/41.0'
    }
```

setting.py 除了配置请求头和文件保存路径之外，还对 ITEM_PIPELINES 添加了 DownloadFlie 类，当程序执行数据存储的时候，会同时执行 ScrapyDownloadPipeline 和 DownloadFlie 类。完成 setting.py 配置后，打开 items.py，定义 Items 类属性，代码如下：

```
import scrapy
class ScrapyDownloadItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
    FileUrl = scrapy.Field()
    FileName = scrapy.Field()
    pass
```

Items 定义了 FileUrl 和 FileName，分别是文件下载链接和文件名。接着在 spiders（文件夹）下新建 download_spider.py 文件，代码如下：

```
from scrapy_download.items import ScrapyDownloadItem
from scrapy.spider import Spider
# 导入 setting.py 配置信息
from scrapy.conf import settings
class downspider(Spider):
    name = "downspider"
    allowed_domains = []
    start_urls = [
        'http://ws.stream.qqmusic.qq.com/C100001IqoFr2rNsGH.
m4a?fromtag=38'
```

```
            ]

        def parse(self, response):
            # 下载方法一
            f = open(settings['FILES_STORE'] + 'MySong.m4a', 'wb')
            f.write(response.body)
            f.close()
            # 下载方法二
            item = ScrapyDownloadItem()
            item['FileName'] = ['PythonBook.zip', 'Python.jpg',
'MyMusic.m4a']
            item['FileUrl'] = [
    'http://d.1.didiwl.com/PYTHON_zryycl.zip',
                              'http://i0.hdslb.com/bfs/archive/9a8f816fdadd
1b814c5ce51e7ead25319166eb92.jpg',
                         'http://ws.stream.qqmusic.qq.com/
C100001IqoFr2rNsGH.m4a?fromtag=38'
    ]
            return item
```

从代码中看到，函数 parse 实现了两种下载方式：

（1）方法一是通过访问文件链接得到链接的响应内容（文件的字节流），然后将响应内容（文件的字节流）写入文件，这种下载方式与 requests 库下载文件一致。

（2）方法二是将文件链接和文件名写入 Items 对象，然后将 Items 传到 Item Pipeline 实现文件下载。

最后在 pipelines.py 实现下载功能，代码如下：

```
from scrapy.pipelines.files import FilesPipeline
from scrapy.pipelines.images import ImagesPipeline
import scrapy
# 导入 setting.py 配置信息
from scrapy.conf import settings

class ScrapyDownloadPipeline(object):
    def process_item(self, item, spider):
```

```
            # 入库处理等操作
            return item

    # 下载功能
    class DownloadFlie(FilesPipeline):
        # 重写get_media_requests
        def get_media_requests(self, item, info):
            for index, url in enumerate(item['FileUrl']):
                yield scrapy.Request(url, meta={'name':
item['FileName'][index]})

        # 重写file_path，设置下载的文件名
        def file_path(self, request, response=None, info=None):
            file_name = settings['FILES_STORE'] + (request.
meta['name'])
            return file_name
```

代码中定义了类 ScrapyDownloadPipeline 和 DownloadFlie：

- ScrapyDownloadPipeline 是在创建项目时自动生成的，在此不做任何处理，但程序依然会执行，因为已被 setting.py 的 ITEM_PIPELINES 激活。
- DownloadFlie 是自定义的类，继承自父类 FilesPipeline（ImagesPipeline），然后将父类的方法 get_media_requests 和 file_path 进行重写。

get_media_requests 的说明如下：

（1）遍历 item['FileUrl']（三个文件下载链接），在 for 循环中使用 enumerate() 获取下载链接所在列表的序号。

（2）获取序号对应 item['FileName'] 的文件名。

（3）scrapy.Request 设置了参数 meta，该参数主要给函数 file_path 传递文件名。

file_path 的说明如下：

（1）settings['FILES_STORE'] 用于获取 setting.py 的路径配置信息，request.meta['name'] 用于获取函数 get_media_requests 的 scrapy.Request() 中的 meta 信息。

（2）如果不重写该方法，Scrapy 就会自行定义文件名，但 Scrapy 对文件命名存在一定缺陷，比如下载链接不规范会出现文件无法保存的情况，如音乐文件链接，若使用 Scrapy 默认文件命名，后缀名会变成 .m4a?fromtag=38。

在 CMD 窗口运行 scrapy_download 项目，程序运行完成后，查看文件下载情况，如图 17-11 所示。



图 17-11 Scrapy 下载文件

17.11 本章小结

Scrapy 是一个为了爬取网站数据、提取结构性数据而编写的应用框架，主要应用在数据挖掘、信息处理或存储历史数据等一系列程序中。其最初是为了页面抓取所设计的，也可以应用在获取 API 所返回的数据（例如 Amazon Associates Web Services）或者通用的网络爬虫中。通过本章的学习，读者应当掌握以下技能：

1. Scrapy 的运行机制

（1）引擎从调度器中取出一个 URL（URL），用于接下来的抓取。

（2）引擎把 URL 封装成请求（Request）传给下载器，下载器把资源下载后封装成应答包（Response）。

（3）爬虫解析 Response。

（4）若解析出实体（Item），则交给实体管道进行进一步的处理。

（5）若解析出的是 URL，则把 URL 交给 Scheduler 等待抓取。

数据抓取的主要目标是从非结构化来源（通常是网页）中提取结构化数据。Scrapy 可以将提取的数据作为 Python 字典返回，但 Python 字典缺乏结构，字典的键会在输入时出现拼写错误或者返回数据不一致，因此，Scrapy 提供了 Items 对象，用于管理和规范爬取数据，使其结构规范化。

2. Item Pipeline 的编写规则

当 Spiders 爬取的数据存放到 Items 之后，回调函数的 return（yield）返回 Items 对象，就会触发 Item Pipeline 对 Items 对象的操作，实现数据存储。Item Pipeline 的编写规则如下：

（1）setting.py 配置数据库信息。数据库信息最好在 setting.py 中配置，这符合统一规范化开发要求。

（2）对 pipelines.py 的类初始化（__init__）函数实现数据库连接。如果使用 SQLAlchemy 入库，还需创建映射类映射数据表。

（3）最后由函数 process_item 实现数据入库。

当抓取网页时，最常见的任务是从 HTML 源码中提取数据，Scrapy 提取数据有一套机制，被称作选择器（Seletors），通过特定的 XPath 或者 CSS 表达式来选择 HTML 中的某部分数据。当然，LXML 和 BeautifulSoup 也可以在 Scrapy 中担任数据清洗的角色。

3. 使用 Files Pipeline 实现下载的步骤

（1）在 Spider 中爬取一个 Item 后，将相应的文件 URL 放入 file_urls 字段中。

（2）Item 被返回之后就会转交给 Item Pipeline。

（3）当这个 Item 到达 FilesPipeline 时，在 file_urls 字段中的 URL 列表会通过标准的 Scrapy 调度器和下载器来调度下载，并且优先级很高，在抓取其他页面前就被处理。而 Item 会一直在这个 Pipeline 中被锁定，直到所有的文件下载完成。

（4）当文件被下载完之后，结果会被赋值给另一个 files 字段。这个字段包含一个关于下载文件的新字典列表，比如下载路径、源地址、文件校验码。files 里面的顺序和 file_url 的顺序是一致的。若下载出错，则不会出现在这个 files 中。

第 18 章

项目实战：Scrapy 爬取 QQ 音乐

18.1 分析说明

在第 13 章，我们介绍了使用 Requests 爬取 QQ 音乐，本章将使用 Scrapy 爬取 QQ 音乐，实现与第 13 章相同的功能，本章沿用第 13 章的爬虫规则实现项目开发。

在歌手列表（https://y.qq.com/portal/singer_list.html）按照字母类别对歌手分类，遍历每个分类下的每位歌手页面，然后获取每位歌手页面下的全部歌曲信息。根据该设计方案列出遍历次数：

（1）遍历每个歌手的歌曲页数。

（2）遍历每个字母分类的每页歌手信息。

（3）遍历每个字母分类的歌手总页数。

（4）遍历 26 个字母分类的歌手列表。

在功能上至少需要实现 4 次遍历，但在实际开发中往往比这个次数要多。统计遍历次数，主要能让开发者对项目开发有整体设计逻辑。

项目开发使用模块化设计思想，对整个项目模块的划分如下：

（1）歌曲下载。

（2）歌手信息和歌曲信息。

（3）字母分类下的歌手列表。

（4）全站歌手列表。

按照上述方案，Scrapy 爬取 QQ 音乐的开发顺序如下。

（1）setting.py：配置爬虫信息，如请求头、数据库信息、文件保存路径。

（2）items.py：定义存储数据对象，主要存储歌曲相关信息。

（3）pipelines.py：数据存储，实现歌曲信息入库和歌曲下载。

（4）spiders 文件夹（music_spider.py）：编写爬虫规则。

由于项目的爬取对象和爬取策略与第 13 章相同，所以本章不再对 QQ 音乐网站进行分析，对爬取网站架构不清晰的读者可阅读第 13 章的有关内容。

18.2 创建项目

首先创建 Scrapy 爬虫项目，命名为 scrapy_music，打开 CMD 窗口，将路径切换到 E 盘，输入创建指令：

```
scrapy startproject scrapy_music
```

完成 Scrapy 项目创建后，在项目中的 spiders 文件夹里创建 music_spider.py 文件，该文件主要实现 Spider 的功能。最后在 PyCharm 中打开项目所在的文件夹，目录结构如图 18-1 所示。

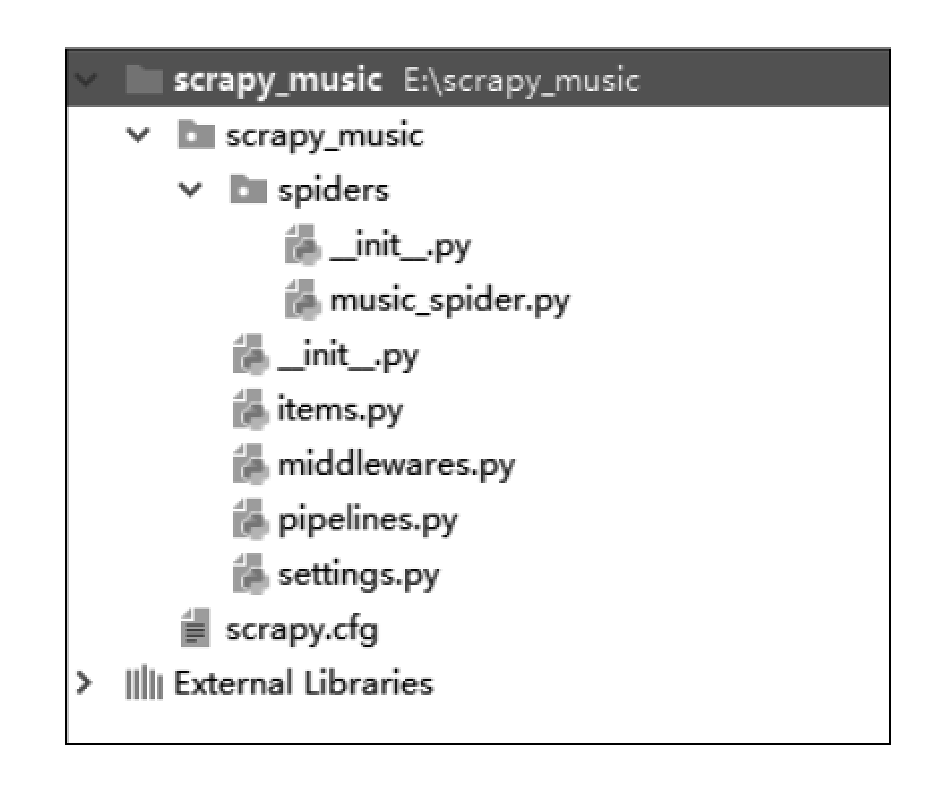


图 18-1 目录结构

18.3 编写 setting

完成项目创建后，接下来就是真正的项目开发。按照制定的开发顺序，首先完成项目配置的开发，打开项目中的 setting.py 文件，由于文件的原始代码较多，因此此处只列出项目所需的代码内容。其代码如下：

```
BOT_NAME = 'scrapy_music'
SPIDER_MODULES = ['scrapy_music.spiders']
NEWSPIDER_MODULE = 'scrapy_music.spiders'
# Obey robots.txt rules
ROBOTSTXT_OBEY = True
# 设置 Item Pipelines
ITEM_PIPELINES = {
   'scrapy_music.pipelines.ScrapyMusicPipeline': 300,
   'scrapy_music.pipelines.DownloadMusicPipeline': 300,
}
# 数据库连接信息
MYSQL_CONNECTION = 'mysql+pymysql://root:1234@localhost:3306/
music_db?charset=utf8'
# 设置请求头
DEFAULT_REQUEST_HEADERS = {
  'Accept': 'text/html,application/xhtml+xml,application/
xml;q=0.9,*/*;q=0.8',
  'User-Agent': 'Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0)
Gecko/20100101 Firefox/41.0',
}
# 设置保存路径
FILES_STORE = 'E:\\full\\'
```

从上述代码看出，项目分别对 Item Pipelines、数据库信息、请求头和文件保存路径进行配置，各个配置说明如下。

- Item Pipelines：创建项目时，默认配置了类 ScrapyMusicPipeline。在此项目中，需要添加一个下载类 DownloadMusicPipeline，该类继承自父类 FilesPipeline，主要实现歌曲下载功能。
- 数据库信息：该配置属于自定义配置信息，变量 MYSQL_CONNECTION 是字符串格式，内容是 SQLAlchemy 连接数据库语句。数据库系统为本地数据库系统，数据库为 music_db。

- 请求头：配置默认的请求头内容，如果项目中发送 HTTP 请求并没有指定请求头，就默认使用该配置作为请求头。
- 文件保存路径：属于自定义配置信息，变量 FILES_STORE 为字符串格式，内容是系统有效路径，主要用于歌曲下载的保存路径。

除此之外，还可以配置并发数和下载延时等相关信息。本节使用默认配置即可，读者可根据以下代码自行配置：

```
    # Configure maximum concurrent requests performed by Scrapy
(default: 16)
    # 设置并发数，Scrapy 默认同一时间可并发 16 个请求
    #CONCURRENT_REQUESTS = 32
    # Configure a delay for requests for the same website (default: 0)
    # See http://scrapy.readthedocs.org/en/latest/topics/settings.
html#download-delay
    # See also autothrottle settings and docs
    # 设置下载延时，延长每个下载之间的时间间隔
    #DOWNLOAD_DELAY = 3
    # The download delay setting will honor only one of:
    # 设置同一域名和同一 IP 的并发数
    #CONCURRENT_REQUESTS_PER_DOMAIN = 16
    #CONCURRENT_REQUESTS_PER_IP = 16
```

18.4 编写 Items

Items 主要用于定义歌曲信息寄存的对象，衔接 Spider 和 Item Pipelines，使两者之间的数据交互传递。

根据第 13 章的数据存储，需要存储的歌曲信息见表 18-1。

表18-1 song数据表

字　段	中文名	字　段	中文名
song_id	主键	song_interval	时长
song_name	歌名	song_songmid	歌曲mid
song_ablum	所属专辑	song_singer	歌手姓名

其中，song_id 是数据表的主键并且数据是自动生成的，在爬取数据中，该数据并不存在，所以可以不用定义。除了上述信息之外，我们还需要添加歌曲下载链接，因为歌曲下载是在 Item Pipelines 实现的，但下载的链接是在 Spider 生成的，所以两者之间需要数据传递。综合分析，打开该项目的 items.py，代码如下：

```
import scrapy
class ScrapyMusicItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
    song_name = scrapy.Field()
    song_ablum = scrapy.Field()
    song_interval = scrapy.Field()
    song_songmid = scrapy.Field()
    song_singer = scrapy.Field()
    song_url = scrapy.Field()
    pass
```

18.5 编写 Item Pipelines

接下来编写 Item Pipelines，该模块主要实现两个功能：歌曲信息入库和歌曲下载。两个功能的数据来源都是 Items 所定义的数据对象。

歌曲信息入库主要由 ScrapyMusicPipeline 实现，该类是创建项目时自动生成的，实现代码如下：

```
# 导入下载类
import scrapy
from scrapy.pipelines.files import FilesPipeline
# 导入 SQLAlchemy
from sqlalchemy import *
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base
# 导入 setting.py 配置信息
from scrapy.conf import settings

# SQLAlchemy 映射数据表
```

```
Base = declarative_base()
class song(Base):
    # 表名
    __tablename__ = 'song'
    # 字段，属性
    song_id = Column(Integer, primary_key=True)
    song_name = Column(String(50))
    song_ablum = Column(String(50))
    song_interval = Column(String(50))
    song_songmid = Column(String(50))
    song_singer = Column(String(50))

# 数据入库
class ScrapyMusicPipeline(object):
    def __init__(self):
        # 获取配置信息 setting.py 的数据库连接
        connection = settings['MYSQL_CONNECTION']
        # 连接数据库
        engine = create_engine(connection, echo=False)
        # 创建会话对象，用于数据表的操作
        DBSession = sessionmaker(bind=engine)
        self.SQLsession = DBSession()
        # 创建数据表
        Base.metadata.create_all(engine)

    def process_item(self, item, spider):
        data = song(
            song_name=item['song_name'],
            song_ablum=item['song_ablum'],
            song_interval=item['song_interval'],
            song_songmid=item['song_songmid'],
            song_singer=item['song_singer'],
        )
        self.SQLsession.add(data)
        self.SQLsession.commit()
        return item
```

上述代码可划分为三部分：库 / 模块导入、SQLAlchemy 映射数据表和歌曲信息入库，各部分说明如下。

- 库 / 模块导入：分别导入 scrapy、FilesPipeline、SQLAlchemy 和 setting.py。
- QLAlchemy 映射数据表：定义 song 类，表名是 song，类属性是数据表的字段。这部分只是定义类映射到数据表，但实际上，数据库和程序还没真正连接。
- 歌曲信息入库：主要由初始方法（__init__）和类方法 process_item 共同实现入库。初始方法（__init__）主要实现程序与数据库的连接，连接方式是使用 SQLAlchemy 实现。类方法 process_item 主要对 Spider 传递的 Items 对象进行入库处理，从数据写入方式来看，Items 对象是每次返回一条歌曲信息，也就是每爬取一首歌曲，就会执行一次歌曲入库和下载。

完成歌曲信息入库后，还要实现歌曲下载，歌曲下载是由 DownloadMusicPipeline 实现的，该类属于自定义类，继承自父类 FilesPipeline，代码如下：

```
# 下载文件
class DownloadMusicPipeline(FilesPipeline):
    # 重写 get_media_requests
    def get_media_requests(self, item, info):
        # 设置文件名
        file_name = item['song_songmid'] + '.m4a'
        yield scrapy.Request(item['song_url'],
                             meta={'name': file_name})

    # 重写 file_path，命名文件名
    def file_path(self, request, response=None, info=None):
        file_name = settings['FILES_STORE'] +
                    (request.meta['name'])
        return file_name
```

歌曲下载由类方法 get_media_requests() 和 file_path() 共同实现，两者都是从父类 FilesPipeline 继承并重写的。父类 FilesPipeline 有一套完善的下载机制，但很多时候并不符合各种各样的下载需求，所以大多数情况下都是通过类的继承和重写的方式实现需求化下载。

- get_media_requests()：每次 Spider 传递的 Items 对象都是一首歌曲的信息，从 Items 对象获取歌曲的 songmid 作为文件名，然后将文件名作为 scrapy.Request 的 meta 参数传递给 file_path()。

- file_path()：接收 get_media_requests() 传递的参数 meta，并读取 setting.py 里面的文件路径配置信息，组合成一个完整的文件路径，最后 DownloadMusicPipeline 将下载的文件以 file_path() 返回值作为文件名。

18.6 编写 Spider

Spider 是整个项目中的难点，同时也是代码量最多的一个功能。根据第 13 章实现的功能发现，整个程序共发送了 6 个不同的请求。在 Spider 中，一个请求代表一个类方法，因此本项目的 Spider 共有 6 个类方法，相应的功能有：

（1）歌手字母分类 A ～ Z。

（2）获取每个字母分类下的每页歌手。

（3）获取每一个歌手信息。

（4）获取歌手的每一页歌曲。

（5）获取每一页的每一首歌曲信息。

（6）每一首歌曲信息。

根据上述分析，实现代码如下：

```
import scrapy
import json
import math
from scrapy_music.items import ScrapyMusicItem
from scrapy.spider import Spider

class QQMusic(Spider):
    name = 'QQMusic'
    allowed_domains = ['qq.com']
    # start_urls 是歌手列表 URL，并使用 %s 设置可控变量
    start_urls = [
        'https://c.y.qq.com/v8/fcg-bin/v8.fcg?channel=singer&page
=list&key=all_all_%s&
         pagesize=100&pagenum=%s&loginUin=0&hostUin=0&format=jsonp'
    ]
```

```
    # 重写 start_requests，遍历歌手字母分类 A ～ Z
    def start_requests(self):
        for i in range(65, 66):
            key = chr(i)
            url = self.start_urls[0] %(key, 1)
            yield scrapy.Request(url, dont_filter=True,
                                 callback=self.get_genre_singer,
                                 meta={'key': key})

    # 获取每个字母分类下的每页歌手
    def get_genre_singer(self, response):
        # 通过参数传递获取字母
        key = response.meta['key']
        # 从函数 start_requests 中得出响应内容，获取总页数
        pagenum = json.loads(str(response.body.decode('utf-8')))
['data']['total_page']
        # 生成列表
        page_list = [x for x in range(pagenum)]
        for p in page_list:
            url = self.start_urls[0] % (key, p+1)
            # dont_filter 取消重复请求
            yield scrapy.Request(url, dont_filter=True,
callback=self.get_singer_songs)

    # 获取每一个歌手信息
    def get_singer_songs(self, response):
        # 获取每个字母分类下的每页歌手的全部信息
        singermid_list = json.loads(response.body.
decode('utf-8'))['data']['list']
        for k in singermid_list:
            url = 'https://c.y.qq.com/v8/fcg-bin/fcg_v8_singer_
track_cp.fcg?loginUin=0&
                  hostUin=0&singermid=%s&order=listen&begin=0&n
um=30&songstatus=1'
                  % (k['Fsinger_mid'])
            yield scrapy.Request(url, dont_filter=True,
                        callback=self.get_singer_info,
```

```
                                  meta={'singermid': k['Fsinger_mid']})

    # 获取歌手的每一页歌曲
    def get_singer_info(self, response):
        # 参数传递获取 singermid
        singermid = response.meta['singermid']
        # 获取歌手的名字、总页数
        singer_info = json.loads(response.body.decode('utf-8'))
        song_singer = singer_info['data']['singer_name']
        songcount = singer_info['data']['total']
        pagecount = math.ceil(int(songcount) / 30)
        for p in range(pagecount):
            url = 'https://c.y.qq.com/v8/fcg-bin/
                         fcg_v8_singer_track_cp.fcg?loginUin=0&
                    hostUin=0&singermid=%s&order=listen&begin=%s&
                    num=30&songstatus=1'% (singermid, p * 30)
            yield scrapy.Request(url, dont_filter=True,
                                 callback=self.get_song_info,
                                 meta={'song_singer': song_singer})

    # 获取每一页的每一首歌曲信息
    def get_song_info(self, response):
        # 参数传递获取歌手名字
        song_singer = response.meta['song_singer']
        music_data = json.loads(response.body.decode('utf-8'))
['data']['list']
        for i in music_data:
            # 设置请求参数
            filename = 'C400' + i['musicData']['songmid']
            # 获取下载歌曲的 vkey
            url = 'https://c.y.qq.com/base/fcgi-bin/fcg_music_
express_mobile3.fcg?loginUin=0&
                    hostUin=0&cid=205361747&uin=0&songmid=%s&filen
ame=%s.m4a&guid=0'
                    % (i['musicData']['songmid'], filename)
            yield scrapy.Request(url, dont_filter=True,
                                  callback=self.get_data,
```

```
                                                    meta={'filename': filename, 'i': i,
                                                    'song_singer': song_singer})

        # 每一首歌曲信息
        def get_data(self, response):
            # 参数传递
            # song_singer 为歌手名字
            # filename 为请求参数
            # i 为歌曲信息
            song_singer = response.meta['song_singer']
            filename = response.meta['filename']
            i = response.meta['i']
            # items.py 文件的类的实例化，用于传递数据给 pipelines.py 实现存储
            items = ScrapyMusicItem()
            # 获取下载歌曲的 vkey
            vkey = json.loads(response.body.decode('utf-8'))['data']
['items'][0]['vkey']
            # 数据写入 items，用于传递数据给 pipelines.py 实现存储
            items['song_url'] = 'http://dl.stream.qqmusic.qq.com/
                 %s.m4a?vkey=%s&guid=0&uin=0&fromtag=66' % (filename,
vkey)
            items['song_singer'] = song_singer
            items['song_name'] = i['musicData']['songname']
            items['song_ablum'] = i['musicData']['albumname']
            items['song_interval'] = i['musicData']['interval']
            items['song_songmid'] = i['musicData']['songmid']
            yield items
```

Spider 共定义了 6 个类方法，分别对应的功能如下。

- start_requests：歌手字母分类 A ～ Z，重写 Spider 的 start_requests。
- get_genre_singer：获取每个字母分类下的每页歌手。
- get_singer_songs：获取每一个歌手信息。
- get_singer_info：获取歌手的每一页歌曲。
- get_song_info：获取每一页的每一首歌曲信息。
- get_data：每一首歌曲信息。

程序运行的时候，start_requests() 获得 start_urls 里面的 URL 信息，然后循环 26 次，分别得到字母 A ～ Z 并将字母传入 URL，生成不同字母分类的 URL。最后通过 scrapy.Request 对 26 个 URL 发送 GET 请求，dont_filter=True 是关闭重复访问 URL 的设置，因为 Scrapy 默认过滤重复访问相同 URL，所以在本项目中我们需要对同一 URL 发送多次请求获取不同的数据；参数 meta 是将 start_requests() 的数据传递到回调函数 get_genre_singer() 中。

- get_genre_singer() 是处理 start_requests() 对 26 个 URL 发送 GET 请求的响应内容。首先分别获取 start_requests() 传递的 meta 参数和当前分类的总页数（来自于 start_requests() 发送 GET 请求的响应内容），使用得到的数据构建新的 URL，其 URL 代表当前分类的每一页的歌手信息，最后对新构建的 URL 发送 GET 请求，回调函数为 get_singer_songs()。
- get_singer_songs() 是处理 get_genre_singer() 发送 GET 请求的响应内容。其响应内容是获取当前分类的每一页的歌手信息，得到的歌手信息是以列表形式表示的，通过遍历该列表分别得到每位歌手的 singermid，然后构建新的 URL，其 URL 代表当前歌手的主页面，如 13.3 节的图 13-5 所示。最后对新的 URL 发送 GET 请求，获取当前歌手的全部信息，回调函数是 get_singer_info()，并传递参数 meta，代表当前歌手的 singermid。
- get_singer_info() 是处理 get_singer_songs() 发送 GET 请求的响应内容，从响应内容中获取歌手的信息并计算歌曲的总页数，使用得到的信息构建新的 URL，代表当前歌手的每一页歌曲，最后对新构建的 URL 发送 GET 请求，回调函数是 get_song_info()，参数 meta 为歌手姓名。
- get_song_info() 是从上一请求的响应内容中获取歌曲信息，歌曲信息以列表形式表示，通过遍历该列表分别获取每一首歌的信息，并使用得到的歌曲信息构建新的 URL，其 URL 用于获取下载歌曲的 vkey 等下载信息，最后对该 URL 发送 GET 请求，回调函数为 get_data()，参数 meta 是歌手歌曲信息。
- get_data() 是最后一个回调函数，主要用于获取歌曲的下载链接和歌曲信息。通过处理上一请求的响应内容得到歌曲下载的信息并构建歌曲下载链接，同时将得到的歌曲信息和新构建的下载链接写入 Items 对象并返回给 Item Pipelines。

从整个 Spider 分析，实现步骤是：全部字母分类歌手列表→当前字母分类歌手列表→当前分类每页的歌手列表→当前分类当前页数的每位歌手→当前歌手的每页歌曲列表→当前歌手的当前歌曲页数的每一首歌曲→获取歌曲信息并下载歌曲。

从实现步骤与第 13 章实现步骤的对比可以发现两者的顺序是相反的，前者是从大到小、从面到点的实现方式；后者是从小到大、从点到面的实现方式。

18.7 本章小结

本章介绍了使用 Scrapy 编写爬取 QQ 音乐的程序，通过本章的学习，读者应当掌握以下技能：

1. Scrapy 爬取 QQ 音乐的开发顺序

- setting.py：配置爬虫信息，如请求头、数据库信息、文件保存路径。
- items.py：定义存储数据对象，主要存储歌曲相关信息。
- pipelines.py：数据存储，实现歌曲信息入库和歌曲下载。
- spiders 文件夹（music_spider.py）：编写爬虫规则。

2. 项目配置

setting.py 是对整个项目的配置，本项目的配置如下：

- Item Pipelines：项目创建时，默认配置了类 ScrapyMusicPipeline。在此项目中，需要添加一个下载类 DownloadMusicPipeline。该类继承自父类 FilesPipeline，主要实现歌曲下载功能。
- 数据库信息：该配置属于自定义配置信息，变量 MYSQL_CONNECTION 是字符串格式，内容是 SQLAlchemy 连接数据库语句。数据库系统为本地数据库系统，数据库为 music_db。
- 请求头：配置默认的请求头内容，如果项目中发送 HTTP 请求时并没有指定请求头，就默认使用该配置作为请求头。
- 文件保存路径：属于自定义配置信息，变量 FILES_STORE 为字符串格式，内容是系统有效路径，主要用于歌曲下载的保存路径。

items.py 主要定义歌曲信息寄存的对象，衔接 Spider 和 Item Pipelines，使两者之间的数据交互传递。

Item Pipelines 主要实现两个功能：歌曲信息入库和歌曲下载。两个功能的数据来源都是 Items 所定义的数据对象。歌曲信息入库主要由 ScrapyMusicPipeline 实现，歌曲下载由类方法 get_media_requests() 和 file_path() 共同实现，两者都是从父类 FilesPipeline 继承并重写的。

3. Spider 的类方法

Spider 共有 6 个类方法，分别说明如下。

- start_requests：歌手字母分类 A-Z，重写 Spider 的 start_requests。
- get_genre_singer：获取每个字母分类下的每页歌手。
- get_singer_songs：获取每一个歌手信息。
- get_singer_info：获取歌手的每一页歌曲。
- get_song_info：获取每一页的每一首歌曲信息。
- get_data：每一首歌曲信息。

从整个 Spider 分析，实现步骤是：全部字母分类歌手列表→当前字母分类歌手列表→当前分类每页的歌手列表→当前分类当前页数的每位歌手→当前歌手的每页歌曲列表→当前歌手的当前歌曲页数的每一首歌曲→获取歌曲信息并下载歌曲。

从实现步骤与第 13 章实现步骤的对比可以发现两者的顺序是相反的，前者是从大到小、从面到点的实现方式；后者是从小到大、从点到面的实现方式。